GENETIC AND EVOLUTIONARY COMPUTATION
Series Editors: David E. Goldberg and John R. Koza

Edited by
Rick Riolo
Terence Soule
Bill Worzel

Genetic Programming Theory and Practice VI

Springer

Rick Riolo, Terence Soule and Bill Worzel (Eds.)

Genetic Programming Theory and Practice VI

Genetic and Evolutionary Computation Series

Series Editors

David E. Goldberg
Consulting Editor
IlliGAL, Dept. of General Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA
Email: deg@uiuc.edu

John R. Koza
Consulting Editor
Medical Informatics
Stanford University
Stanford, CA 94305-5479 USA
Email: john@johnkoza.com

Selected titles from this series:

Markus Brameier, Wolfgang Banzhaf
Linear Genetic Programming, 2007
ISBN 978-0-387-31029-9

Nikolay Y. Nikolaev, Hitoshi Iba
Adaptive Learning of Polynomial Networks, 2006
ISBN 978-0-387-31239-2

Tetsuya Higuchi, Yong Liu, Xin Yao
Evolvable Hardware, 2006
ISBN 978-0-387-24386-3

David E. Goldberg
The Design of Innovation: Lessons from and for Competent Genetic Algorithms, 2002
ISBN 978-1-4020-7098-3

John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, Guido Lanza
Genetic Programming IV: Routine Human-Computer Machine Intelligence
ISBN: 978-1-4020-7446-2 (hardcover), 2003; ISBN: 978-0-387-25067-0 (softcover), 2005

Carlos A. Coello Coello, David A. Van Veldhuizen, Gary B. Lamont
Evolutionary Algorithms for Solving Multi-Objective Problems
ISBN: 978-0-306-46762-2*,* 2002; ISBN 978-0-387-33254-3 (2nd Edition), 2007

Lee Spector
Automatic Quantum Computer Programming: A Genetic Programming Approach
ISBN: 978-1-4020-7894-1 (hardcover), 2004; ISBN 978-0-387-36496-4 (softcover), 2007

William B. Langdon
Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming! 1998
ISBN: 978-0-7923-8135-8

For a complete listing of books in this series, go to http://www.springer.com

Rick Riolo
Terence Soule
Bill Worzel
(Eds.)

Genetic Programming Theory and Practice VI

Springer

Rick Riolo
Center for the Study of Complex Systems
323 West Hall
University of Michigan
Ann Arbor, MI 48109
rlriolo@umich.edu

Terence Soule
Department of Computer Science
University of Idaho
Janssen Engineering Building
Moscow, ID 83844-1010
tsoule@cs.uidaho.edu

Bill Worzel
Genetics Squared
401 W. Morgan Rd.
Ann Arbor, MI 48108
billw@genetics2.com

ISSN: 1932-0167
ISBN: 978-0-387-87622-1
e-ISBN: 978-0-387-87623-8
DOI: 10.1007/978-0-387-87623-8

Library of Congress Control Number: 2008936134

© 2009 Springer Science+Business Media, LLC
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

springer.com

Contents

Contributing Authors

Mostafa Z. Ali is an Assistant Professor of Computer Science at Jordan University of Science and Technology, Irbed, Jordan (java.duke@yahoo.com).

Arpit Arvindkumar Almal is an evolutionary engineer at Genetics Squared, Inc., a computational discovery company (aalmal@umich.edu).

Peter C. Andrews is a software engineer in the Computational Genetics Laboratory at Dartmouth Medical School (Peter.C.Andrews@Dartmouth.edu).

Mohamed B. Bader-El-Den is a PhD candidate in the Department of Computing and Electronic Systems at University of Essex (mbbade@essex.ac.uk).

Wolfgang Banzhaf is a professor and chair of the Department of Computer Science at Memorial University of Newfoundland, St. John's, NL, Canada (simonh@cs.mun.ca).

Ying Becker is Vice President, Head of Global Quantitative Active Equity Research, Advanced Research Center at State Street Global Advisors, Boston, MA USA (Ying_Becker@ssga.com).

Stuart W. Card is a PhD candidate in the Department of Electrical Engineering and Computer Science at Syracuse University(cards@ntcnet.com).

Xiangdong Che is a PhD candidate in the Department of of Computer Science at Wayne State University, Detroit, MI, U.S.A (sean_che@wayne.edu).

Shu-Heng Chen is a Professor of Economics at the National Chengchi University, Taiwan (chchen@nccu.edu.tw).

Peng Fei is a senior quantitative research analyst for the Advanced Research Center at State Street Global Advisors (SSgA), the investment management arm of State Street Corporation, Boston, MA (peng_fei@ssga.com).

Georges Gielen is a Professor of Electrical Engineering Katholieke Universiteit Leuven, Belgium (georges.gielen@esat.kuleuven.be).

Casey S. Greene is a graduate student in the Computational Genetics Laboratory at Dartmouth Medical School (Casey.S.Greene@Dartmouth.edu).

Simon Harding is a postdoctoral fellow in the Department of Computer Science at Memorial University of Newfoundland, St. John's, NL, Canada (simonh@cs.mun.ca).

Malcolm Heywood is a Professor of Computer Science at Dalhousie University, Halifax, NS, Canada (mheywood@cs.dal.ca).

Minkyu Kim is a doctoral candidate in the Department of Electrical Engineering and Computer Science at Massachusetts Institute of Technology (minkyu@mit.edu).

Jon Klein is a Senior Research Fellow at Hampshire College in Amherst, Massachusetts (jk@artificial.com).

Michael F. Korns is Chief Technology Officer at Investment Science Corporation (mkorns@korns.com).

Mark E. Kotanchek is Chief Technology Officer of Evolved Analytics, a data modeling consulting and systems company (mark@evolved-analytics.com).

W. B. Langdon is a Research Officer jointly in the Departments of Mathematical, Biological Sciences and Computing and Electronic Systems of the University of Essex (wlangdon@essex.ac.uk).

Duncan MacLean is co-founder of Genetics Squared, Inc., a computational discovery company working in the pharmaceutical industry (dmaclean@acm.org).

Trent McConaghy is Chief Scientific Officer at Solido Design Automation Inc., as well as PhD candidate at ESAT-MICAS, Katholieke Universiteit Leuven, Belgium (trent_mcconaghy@yahoo.com).

Andrew McIntyre is a Post Doctoral Researcher in the Faculty of Computer Science at Dalhousie University, Halifax, NS, Canada (armcnty@cs.dal.ca).

Nic McPhee is a Professor of Computer Science at the University of Minnesota, Morris (mcphee@morris.umn.edu).

Chilukuri K. Mohan is Professor of Electrical Engineering and Computer Science at Syracuse Universit (ckmohan@syr.edu).

Jason H. Moore is the Frank Lane Research Scholar in Computational Genetics and Associate Professor of Genetics at Dartmouth Medical School (Jason.H.Moore@Dartmouth.edu).

Loryfel Nunez is a Research Scientist at Investment Science Corporation (lnunez@investmentsciencecorp.com).

Una-May O'Reilly is a Principal Research Scientist in the Computer Science and Artificial Intelligence Laboratory at Massachusetts Institute of Technology (unamay@csail.mit.edu).

Pieter Palmers is a PhD candidate at the Microelectronics and Sensors (ESAT-MICAS) Lab at Katholieke Universiteit Leuven, Belgium (pieter.palmers@esat.kuleuven.be).

Riccardo Poli is a Professor of Computer Science in the Department of Computing and Electronic Systems of the University of Essex, UK (rpoli@essex.ac.uk).

Robert G. Reynolds is a Professor of Computer Science at Wayne State University, Detroit, MI, U.S.A (reynolds@cs.wayne.edu).

Rick Riolo is Director of the Computer Lab and Associate Research Scientist in the Center for the Study of Complex Systems at the University of Michigan (rlriolo@umich.edu).

Guido F. Smits is a Research and Development Leader in the New Products Group within the Core R&D Organization of the Dow Chemical Company (gfsmits@dow.com).

Terence Soule is an Associate Professor of Computer Science at the University of Idaho, Moscow, ID (tsoule@cs.uidaho.edu).

Lee Spector is a Professor of Computer Science in the School of Cognitive Science at Hampshire College in Amherst, Massachusetts (lspector@hampshire.edu).

Michiel Steyaert is a Professor of Electrical Engineering Katholieke Universiteit Leuven, Belgium (michiel.steyaert@esat.kuleuven.be).

Leonardo Vanneschi is an Assistant Professor of Computer Science at the University of Milano-Bicocca, Milan, Italy (vanneschi@disco.unimib.it).

Ekaterina Vladislavleva is a postdoctoral researcher in the Computer Arithmetics and Numerical Techniques Group in the Department of Mathematics and Computer Science, Antwerp University, Belgium (katya@vanillamodeling.com).

Bill C. White is a Senior Programmer in the Computational Genetics Laboratory at Dartmouth Medical School (bill.c.white@Dartmouth.edu).

Garnett Wilson is a postdoctoral fellow in the Department of Computer Science at Memorial University of Newfoundland, St. John's, NL, Canada (gwilson@cs.mun.ca).

Bill Worzel is the Chief Executive Officer and co–founder of Genetics Squared, Inc., a computational discovery company working in the biotech industry to develop molecular diagnostics (billw@genetics2.com).

Tina Yu is an Associate Professor of Computer Science at the Memorial University of Newfoundland, Canada (tinayu@cs.mun.ca).

Ren-Jie Zeng is an Assistant Research Fellow at the AI-ECON Research Center, National Chengchi University, Taiwan (93258038@nccu.edu.tw).

Preface

The work described in this book was first presented at the Sixth Workshop on Genetic Programming, Theory and Practice, organized by the Center for the Study of Complex Systems at the University of Michigan, Ann Arbor, 15-17 May 2008. The goal of this workshop series is to promote the exchange of research results and ideas between those who focus on Genetic Programming (GP) theory and those who focus on the application of GP to various real-world problems. In order to facilitate these interactions, the number of talks and participants was small and the time for discussion was large. Further, participants were asked to review each other's chapters *before* the workshop. Those reviewer comments, as well as discussion at the workshop, are reflected in the chapters presented in this book. Additional information about the workshop, addendums to chapters, and a site for continuing discussions by participants and by others can be found at http://cscs.umich.edu/gptp-workshops/gptp2008.

We thank all the workshop participants for making the workshop an exciting and productive three days. In particular we thank the authors, without whose hard work and creative talents, neither the workshop nor the book would be possible. We also thank our keynote speakers Josh Bongard, Assistant Professor in the Department of Computer Science, University of Vermont, and Marc Schoenauer, Senior Researcher (Directeur de Recherche) with INRIA, the French National Institute for Research in Computer Science and Control, and Editor in Chief of the Evolutionary Computation Journal. Bongard (along with Cornell graduate student Mike Schmidt) described their work (with Hod Lipson) on using GP to reverse engineer coupled, nonlinear dynamical systems, work that led to a "Humie" (Human-Competitive Award) honorable mention at GECCO-2007. Schoenauer described why all kinds of Evolutionary Algorithms are not more widely used for "real world" applications, and presented some suggestions for how to increase their reach. Both talks inspired a great deal of discussion among the participants throughout the workshop.

The workshop received support from these sources:

- The Center for the Study of Complex Systems (CSCS);
- Third Millennium Venture Capital Limited;
- Michael Korns, Investment Science Corporation.
- State Street Global Advisors, Boston, MA; and
- Biocomputing and Developmental Systems Group, Computer Science and Information Systems, University of Limerick.

Thanks also to William & Barbara Tozier of Vague Innovation LLC for arranging and subsidizing the "open-format evening session" at the Arbor Brewing Company.

We thank all of our sponsors for their kind and generous support for the workshop and GP research in general.

A number of people made key contributions to running the workshop and assisting the attendees while they were in Ann Arbor. Foremost among them was Howard Oishi. After the workshop, many people provided invaluable assistance in producing this book. Special thanks go to Sarah Chernng, who did a wonderful job working with the authors, editors and publishers to get the book completed very quickly. Thanks to William Tozier for assisting in copy-editing some of the chapters. Valerie Schofield and Melissa Fearon's editorial efforts were invaluable from the initial plans for the book through its final publication. Thanks also to Deborah Doherty of Springer for helping with various technical publishing issues. Finally, we thank Carl Simon, Director of CSCS, for his support for this endeavor from its very inception.

RICK RIOLO, TERENCE SOULE AND BILL WORZEL

Foreword

Genetic programming has now been practiced, in one form or another, for around twenty years, and one might worry that the field is old enough to begin stagnating, giving way to newer approaches and sharper tools. Participation in one of the annual Genetic Programming Theory and Practice Workshops is enough to disabuse anyone of such a notion very rapidly. This three-day workshop, organized annually by the Center for Study of Complex Systems (CSCS) at the University of Michigan, was held for the sixth time, in Ann Arbor, in May, 2008. The new ideas presented, the old ideas reconsidered, and the intensity of the interactions were characteristic of a field that is still changing quickly, and in which significant advances are frequent. This volume includes the papers presented there, and they represent many excellent contributions by people working at the frontiers of the field. But no printed work can capture the energy that manifests itself at these workshops, where the most renowned GP theorists are battered for new results by the leading practitioners in GP application to real-world problems. Practitioners must frequently make decisions about problem representation, choice of algorithms, setting of parameters, and related issues, with much less guidance from GP theory than they would like. They are sometimes frustrated trying to explain why a particular approach works so well on one problem, and so poorly on another. How should problems be classified so that GP theory would guide the practitioner in making those choices? The GP theorists, in turn, look to the practitioners to identify the critical shortcomings of current theory, and to provide challenging questions for new theory to explain. This year's theory side included lively discussions of application of information theory in GP, evolutionary dynamics in abstract systems vastly simplified from GP, biasing of fitness rewards toward the cases that have been hard to fit, sampling biases, effects of elitism on bloat, and work in multi-objective GP. On the more applied side, participants looked at stock market and trading strategy applications, evolving strategies for games, and lots of work on symbolic regression/classification problems. This year, there was even a paper on hardware–about doing GP on graphics processing units!

Most of the participants write their own GP algorithms, often mixing the roles of theoretician and practitioner as they create new approaches to solve problems—illuminated by theory. Or they exercise the evolutionary process so that new theory can be formulated based on success or failure in solving particular problems. Each year, we hear from participants who have taken home some ideas from the previous workshop, have put them to work in their own problem domains, and have brought back results that may either support or undercut the generality of the approaches. Sometimes we manage to articulate

that two purportedly different approaches share the same key to success, and we end up with a nugget we can all take home to test for ourselves. Sometimes, as this year, a consensus emerges that we've been missing something—leaving something on the table—that should have been a "no-brainer." When consensus is not so easy, it is often fostered by heated discussions—okay, not heated according to a physics meeting's standards, perhaps, but heated for a bunch of computer geeks! Often, we each leave with a resolution to find out for ourselves if something will work, whether the idea was suggested by someone else or perhaps we stumbled upon it during the course of the debate. My own notebook is full of "to do" items at the end of each workshop, and I have found many of the ideas to be well worth pursuing. I come to the workshops as both an academic, presenting research work with my students, and as an industry practitioner, creating commercial evolutionary design software for solving practical engineering problems, and what I learn is helpful in both roles. The workshop is a great forum for presenting new work—with at least forty minutes to present, and the attention of many of the strongest innovators in the field, plus ample time for discussion after each pair of papers, authors get excellent feedback on their work and on interesting directions for next steps. The organizers have managed to bring in "new blood" each year, while continuing to attract the leaders who have participated many times. It's a welcoming environment. No one would be there if not already recognized as an expert in genetic programming, so there's no need for defensiveness in the discussions—people say what's on their minds, and if they sometimes say something that sounds overly simplistic, it's not a problem; there's not much penalty for thinking on one's feet.

The organizers of the sixth GPTP workshop have once again done an excellent job. Particular thanks are due to Rick Riolo (University of Michigan), Bill Worzel (Genetics Squared), and Terry Soule (University of Idaho). As always, Howard Oishi of the CSCS did a great job with the logistics. I've had the pleasure of participating since the beginning of the workshops, and I regard it as the most intense and concentrated source of good ideas in evolutionary computation that I regularly participate in. Even though we allow lengthy discussion periods, our emcee Bill always ends up having to shut off the discussion. But it continues in the evenings over dinner and drinks, although the topics there sometimes extend well beyond either the theory or practice of GP.

Have a look at these papers for some excellent contributions to the field, and if you have interesting GP theory or practice work, consider this workshop as a good place to present it!

Erik Goodman,
Professor, ECE, Michigan State University
VP Technology, Red Cedar Technology, Inc.
July, 2008

Chapter 1

GENETIC PROGRAMMING: THEORY AND PRACTICE

An Introduction to Volume VI

Terence Soule[1], Rick L. Riolo[2] and Una-May O'Reilly[3]

[1]*University of Idaho, Moscow, ID;* [2]*Center for the Study of Complex Systems, University of Michigan, Ann Arbor, MI;* [3]*Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.*

Abstract The annual "Genetic Programming: Theory and Practice" workshop brings together leading theoreticians and practitioners to help bridge the gap between theory and practice in the field of genetic programming (GP). The general goal of this workshop is to advance GP so that it can reliably be used to solve industrial scale problems without requiring excessive amounts of GP specific expertise. One of the most important outcomes of the workshop is the identification of active research themes that the participants feel are crucial to this goal. These are the themes that were independently repeated throughout the research presentations and that became apparent spontaneously during the associated discussions. This chapter summarizes those themes and relates them to the results presented individual papers in the expectation that other researchers may use them as a guide for the direction of their own research projects.

1. The Workshop

The first Genetic Programming: Theory and Practice workshop was organized in 2003 by the Center for Complex Studies (CSCS) of the University of Michigan to bring together practitioners and theorists to bridge the gap between what challenges practitioners were facing and the questions theorists were studying. The goal of the workshop is still to bring together practitioners and researchers, but the field has changed significantly since that first workshop. In the first volume genetic programming was described as "a young art that is just beginning to make its way into applications programming in a serious way" (Riolo and Worzel, 2004). Since then GP has made significant advances

from art to craft, and has generated a number of very significant successes (Koza et al., 2003; Koza et al., 2004; Lenahan, 2006; Tan et al., 2006) so that its value for solving complex problems is well established. Thus, the major themes that arose in the sixth "Genetic Programming: Theory and Practice" workshop emphasize how to improve and broaden GP's capabilities, rather than how to demonstrate them.

GP's progress from a young art toward a mature engineering displine was illustrated by Marc Schoenauer's keynote presentation, titled "Crossing the Chasm — Lessons from the continuous case." His not entirely tongue-in-cheek goal as an evolutionary researcher is "to become extinct" by supplying GP novices with the tools to apply GP to practical problems without the need for a GP expert to design the system. This has been a repeated theme at recent workshops with many attendees advocating the creation of an easy to use, "turn-key" GP system for non-experts and several projects pursuing such a system (Moore et al., 2007). Schoenauer's efforts have been directed towards the GUIDE (Graphical User Interface for Dream Experiments) system (Collet and Schoenauer, 2003).

As in previous years the workshop was held on the campus of the University of Michigan in Ann Arbor, Michigan. This year's workshop was supported by Third Millennium, State Street Global Advisors (SSgA), Investment Science Corporation (ISC), and the Biocomputing and Developmental Systems Group of the University of Limerick. Although the workshop brings together leaders from the cutting edge of both theory and practice, the results of the workshop are of use to even a relative novice in the field of GP. With that in mind we begin this chapter with a brief summary of GP for readers with only a passing familiarity with it. The summary is followed by a discussion of the major themes that arose during the workshop and how the presented research relates to those themes. These themes represent the research areas that the workshop participants identified as crucial to advancing GP state of art. We hope they will serve as a guide and inspiration to other researchers.

2. A Brief Introduction to Genetic Programming

Genetic programming (GP) is an automatic programming technique modeled on natural evolution–see (Koza, 1992; Koza, 1994; Banzhaf et al., 1998a; Langdon and Poli, 2002) for a detailed introduction. It begins with a *population* of *individuals* that represent potential solutions to a user defined problem. A *fitness function* is used to assign each individual a fitness based on how well it solves the problem. Individuals are *selected* to act as parents based on their fitness. Selected individuals typically undergo some form of *crossover*, which generates new offspring; individuals that are a combination of two (or more) parent

individuals. These offspring individuals undergo *mutation* independently and then are used to generate a new, next generation, population.

GP is a kind of genetic or evolutionary algorithm. It tends to be distinguished from other evolutionary algorithms by the fact that the potential solutions are represented as computationally *executable* structures such as LISP expression trees (Koza, 1992; Wu and Banzhaf, 1998), programs for stack or register based machines (Kantschik and Banzhaf, 2002; Spector and Robinson, 2002; Robinson, 2001), program grammars (Ryan et al., 2002; McConaghy et al., 2007), or specially wrapped procedures. The result of executing the solution yields a side effect or output that is the input to the fitness function.

The processes of selection, crossover, and mutation are applied repeatedly in a simplified version of evolution under artificial selection. Over time the continual selection of more fit individuals combined with the generation of new individuals that are recombinations of previous individuals with above average fitness leads to improving individuals and the 'evolution' of successful solutions. Although the basic GP algorithm is conceptually simple, a number of more advanced, and more complex, techniques have become increasingly common, particularly among practitioners dealing with industrial scale, real world problems. These commonly used, advanced techniques are described in the discussion section of this chapter.

By evolving executable structures, GP can be applied to a significantly wider range of applications than can GAs, but the GP approach also introduces a number of complications. First, because the individuals are executable structures, its representations and/or variation operators (crossover and mutation) must be designed to exclusively create syntactically valid individuals. Second, the behaviour and ensuing output of executable structures is typically very sensitive to even extremely small semantic changes, which makes the search space extremely complex. Finally, because the individuals have variable, and evolving, sizes, there is a tendency for them to "bloat"—to grow rapidly without a corresponding increase in size. This bloat can be a significant drain on memory and computational power and thus controlling it is useful for efficiency sake alone. How the large structures that result from bloat actually behave is also hard for people to understand, which is undesirable for applications in which humans are required to look and and evaluate the candidate solutions (as described later in this chapter). Note that the above complicating factors have significant roles to play in the major themes that arose at the workshop and which are described in this chapter.

Since its introduction in the early 1990's GP has had a number of signature successes. GP's most significant achievements have been in two broad areas: creating novel engineering solutions, for example novel antenna designs (Lohn et al., 2005) and analog circuit designs (Peng et al., 2007; McConaghy et al., 2007), and in tackling industrial scale modeling, data analysis, search and opti-

mization problems (Korns, 2006; Becker et al., 2006; Kotanchek et al., 2006). These are application areas that are typically the focus of intense human effort, because

1. traditional analytic techniques fail or cannot be applied,
2. there are no solutions based on "first principles" and
3. there can be a big payoff from even small increases in performance.

Further, GP's successes in these areas illustrate its most important, and perhaps unique, strength – GP's ability to generate *innovative* and *insightful* solutions, something that has proven difficult for other search and optimization techniques.

However, these successes have been sporadic, often requiring significant input from researchers experienced with GP in order to achieve them. Thus, a major goal of GP research is to determine the critical factors that will make GP's successes more reliable while requiring less GP expertise. The themes that arose during the workshop are ones whose pursuit is expected to achieve this end.

3. Key Themes for Achieving Reliable, Industrial Scale GP

A large number of topics were discussed in varying detail during the workshop. Here we present the most significant themes identified by participants as being critical to achieving reliable, industrial scale GP:

- Making efficient and effective use of test data: In most practical applications there are significant limitations in the data that the GP must learn from. On the one hand are the "high D low N problems" such as microrarray analysis. For such problems there may be tens of thousands of variables, the majority of which are irrelevant, and only a few hundred, or less, data points to work from. On the other extreme are problem domains which have huge numbers of training points, e.g., for financial modeling, and a relatively low dimensionality (dozens of variables). For all domains, maximizing GP performance requires leveraging the training data, a key topic in a number of the papers.

- Sustaining the long term evolvability of our GP systems: GP, like most heuristic search/optimization techniques, runs a risk of converging on a locally, rather than a globally, optimal solution. By reliably extending the evolutionary process in a way that successfully explores relatively unexplored regions of the search space (as opposed to wandering in the small local neighborhood) significant advances are possible.

- Exploiting discovered subsolutions for reuse: GP constantly is constantly creating new structures and innovations. However, the value of given

structure may not be immediately apparent. Thus by allowing the GP process to 'set aside' sub-solutions for later reuse, significant improvements are possible.

- Increasing the role of a Domain Expert: GP's performance can be leveraged by including subject domain expert knowledge, either in advance to improve the search process, or post evolution to interpret and vet evolved solutions. Examples of both approaches, often in the same research, appeared throughout the workshop.

One of the most interesting aspects of these themes is the degree to which they are inter-dependent. For example, extended evolution requires that the test data be used in way that avoids being trapped in local optima; in an "extended" evolutionary run partial solutions from early generations can be reused effectively in later generations; expert knowledge can be used to identify key features of a test set, increasing its usefulness or to identify where subsolution reuse is potentially advantageous. Thus, while in the following sections we attempt to organize and summarize the presented research based on how it contributed to or addressed these themes in isolation, it should be acknowledged that the inter-relationships are much more significant than our short summaries imply. Therefore the reader is doubly encouraged to read the full papers.

Using the Data

In many problems, particularly real-world applications, there are shortcomings in the test data that the GP learns from. Typical shortcomings include test data that over-represents some cases over others, test data that is too extensive to use exhaustively (due to the cost of fitness evaluations), and test data that is limited due to the cost to collect it. In all of these cases maximizing the GP's performance requires leveraging the data by focusing the search on the test cases that are most relevant. A number of different techniques were presented depending on the primary limitations of the data.

Some researchers addressed problems with potentially millions of training points. Practical limitations often make it impossible to train on such large data sets, but random subsets of the data may miss critical features of the problem domain. Thus, more focused methods of data selection, particularly methods that identify critical features of the data, are necessary.

In the first day's keynote address Bongard and Schmidt presented an "active probing" technique designed to identify a small subset of test cases that disambiguated between evolving candidate models. Their goal was to find coupled, non-linear equations (candidate models) that accurately model multi-variable, non-linear, dynamical systems. Because they dealt with real-valued systems they effectively had an infinite set of test cases to train on. Thus, they needed to pick a subset or subsets for training. However, with a fixed set of points,

particularly a random set, two very different models often returned similar or identical rankings (fitnesses). To overcome this limitation they introduced a co-evolving population of test points. Individuals in this second population were selected based on their ability to disambiguate candidate models. They found that the second population quickly evolved to identify the bounds between basins of attraction in the dynamical system, thereby maximizing the observed differences between models and significantly improving the overall results without requiring excessive training points.

Similarly, in order to scale their GP system to large data sets, sets too large to be used exhaustively, McIntyre and Heywood (Chapter 4) also used competitive co-evolution of the training points. In their case the individuals in the population of training points were selected based on the ability of the points to distinguish between Pareto non-dominated learners. As with Bongard and Schmidt this allowed the GP to distinguish between solutions that would potentially be ranked equivilently on a randomly selected training set.

In a more generic approach Korns and Nunez (Chapter 14) use *vertical slicing* to select a subset of points for training on a problems with up to 30 variables, tens of thousands of training cases, and significant levels of noise. In vertical slicing data points are sorted by dependent variable and then every Nth case is used. This gives a fairly uniform subset for training. Even with this reduction they also used significant code optimization and very large cluster computing.

Finally, for very large data sets, researchers can simply rely on increased efficiency to allow them to use the entire set. In Chapter 15 Banzhaf et al. survey current approaches for running GP on general purpose graphics processor units (GPGPUs). GPGPUs have several features for improving their performance: a simple pipeline architecture, a high degree of parallelism, and memory that is very close to the processor. Taking advantage of these features in a GP can be difficult, but they show that efficient implementations of GP on a GPGPU can lead to speed-ups of $1000x$ or more. This makes exhaustive use of very large data sets much more practical.

Even when the test data can be used exhaustively the GP may focus on a simple sub-problem. This is particularly likely if the data set is very large, with relatively few cases representing the more difficult portions of the problem. In Chapter 5 Klein and Spector introduce 'Historically Assessed Hardness' (HAH) as an approach to 'refocus' GP search on the harder cases. In HAH the fitness value of individual test cases are adjusted according to how often or how well they were solved previously. For example, if a particular training case was solved by only a few individuals its relative fitness value would increase and if it was solved by a majority of individuals its fitness value would decrease. They examined a number of reweighting schemes and reweighting histories (previous generation, previous run, etc.). In general they found that the method was quite robust to the method of rewieghting and choice of historical window

and generally led to significant improvements in performance for a minimal computational cost.

Kotanchek et al. (Chapter 10) presented results for an industrial problem with two critical restrictions: collecting test data is expensive and results must be trustworthy. To address trustworthiness they evolved ensembles of models so that incorrect results by one ensemble member are likely to be covered by other members of the ensemble. To address the cost of collecting test data they introduced 'adaptive design of experiments' (adaptive DOE). The adaptive design of experiments model begins by evolving an ensemble on a small set of test data. The points of maximum divergence between the models in the ensemble are identified; these are the points of minimal trustworthiness. Additional test data is collected at those points and this additional data is used to retrain the models. Adaptive DOE has two significant advantages. First, it minimizes the amount of data that needs to be collected, by focusing data collection specifically on regions where the models diverge. Second, it increases confidence in the models by focusing learning on the points where the models diverge, which is where the models are least trustworthy.

In Chapter 11 Bader-El-Den and Poli introduce an incremental approach to solving SAT problems. SAT requires determining whether there is a set of variable assignments that makes a Boolean expression (written in conjunctive normal form) true. Each clause in the expression can be viewed as an additional data point that must be considered. Thus, the full expression effectively has too many cases/clauses to solve directly. Their incremental algorithm begins with a limited version of the full SAT problem (i.e. a limited number of clauses to solve). Depending on performance in the previous generation additional clauses are added or removed from the expression currently being tested, until the full expression is solved. This incremental approach allowed GP to solve SAT problems that were otherwise unsolvable.

Often a data set will contain irrelevant and misleading variables. These can mislead the GP system or cause overfitting, in which case the GP evolves solutions that use spurious correlations. In Chapter 3, Card and Mohan used Mutual Information *sufficiency* to rank the relevancy of inputs in a set of noisy symbolic regression problems. Their critical advance is creating an efficient and effective method for discretizing continuous valued data so that Shannon's Entropy can be calculated and used in the ranking process. The results show that the approach significantly outperforms linear correlation and, perhaps more importantly, generates a threshold that differentiates between relevant and irrelevant inputs.

Note that several of the approaches to using data efficiently, e.g., co-evolution, HAH and adaptive DOE, basically are all methods to focus the evolutionary process on particular parts of the problem space. For example, HAH uses an explicit re-weighting procedure to focus attention on particular data points, a

process co-evolution achieves by selectively reproducing (with variation) problematic data points.

Extended Evolution

A second general theme was the need to extend the evolutionary runs, preferably indefinitely, while suffering no loss of evolvability. Premature convergence, the tendency of evolving populations to converge on a sub-optimal solution, has been a long standing problem in all branches of evolutionary computation. However, previous research has focused on how to direct the quickly converging process toward discovering sufficiently high quality solutions in the given time span according to some pre-decided budget of computation. In contrast, in this year's GPTP workshop many participants expressed the opinion that GP systems should be designed to run indefinitely, continually improving upon solution quality .

There are a number of factors driving this new goal. First, the difficulty of the problems being addressed by GP has increased by several order of magnitude, so there is no longer an expectation that any *a priori* identifiable amount of time will necessarily find a globally optimal solution. Instead participants envisioned systems that run continuously and that are polled for the current best solution when necessary. Second, many of the current applications are dynamic, for example, choosing investments in changing markets or monitoring and optimizing on-going physical/chemical processes. Thus, there is no fixed optimal solution. Third, many participants were interested in leveraging previous results, e.g. through the code reuse and building hierarchical solutions. This requires on-going evolution in which previous results are "folded into" the next round of evolution. A final reason for the interest in extended, "open-ended" evolution may be a result of the challenge put forward during a keynote address last year, at GPTP-2007 (Soule et al., 2007), in which Banzhaf (Banzhaf et al., 2006) argued that GP should begin to use more ideas from biological evolution, one of which is the apparent never-ending aspect of biological evolution.

What can be done to run GP profitably over a **long** scale time? Here some of the individual contributions to the workshop that are previously cited in Section 3.0 speak again:

- control premature convergence with co-evolution (Bongard and Lipson, 2007).

- control premature convergence with endogenous system-individual adaptation (Moore et al., Chapter 9 and Kotanchek et al., Chapter 10).

- use metrics or GP run behaviour to control elements of the algorithm (Klein and Spector's HAH, Chapter 5 and Card and Mohan, Chapter 3).

- control the cost of fitness evaluation so GP is practical via fitness estimation (Bongard and Lipson, 2007).
- address the cost of fitness evaluation through algorithm/software optimization (Korns and Nunez, Chapter 14, and McConaghy et al., Chapter 8, who choose algorithms with care and profiles them for expense) or faster hardware (Korns and Nunez, Chapter 14 and Banzhaf et al., Chapter 15, and Moore et al., Chapter 9).
- control bloat—almost every paper in this volume uses some, or several, methods to control bloat, and two chapters address the theoretical causes of bloat (Poli et al., Chapter 7 and Soule, Chapter 6).
- create temporal system-level decomposition to sustain evolvability (e.g. Kotanchek et al., Chapter 10 and McConaghy et al., Chapter 8, and Korns and Nunez, Chapter 14).
- use every individual in the population to maximize the benefit of extended, or unextended, evolution (e.g. McIntyre and Heywood, Chapter 4).

In general, several specific obstacles to extended evolution were identified and discussed throughout the workshop. First, premature convergence must still be avoided. Second, code bloat must be controlled. Third, metrics to measure progress must be developed.

It should be noted that extended evolution was not a universally embraced goal. For some domains, external factors impose a fixed deadline or computational buget; e.g. a solution is required by the opening of the financial markets or must be computed on a cluster with allocated resources. In these cases extending evolution serves little purpose. On the other hand, a number of participants also noted the goals of maintaining diversity and avoiding bloat can be useful even if indefinitely extended evolution is not pursued.

Avoiding Premature convergence in Extended Evolution. A number of techniques were presented that could avoid premature convergence, or indeed convergence at all. Some of these techniques were devised specifically to target convergence; others are simply likely to have convergence avoidance as an additional outcome.

First, many of the techniques described previously for optimizing the use of test data result in dynamic fitness functions, which are known to limit convergence. Bongard's active probing, McIntyre and Heywood's competitive co-evolution, Klein and Spector's HAH, Konetchek et al's adaptive DOE, and Bader-El-Den and Poli's incremental SAT solver all result in continuous changes to either the training points or the relative weights of the training points, which creates a dynamic fitness function. Thus, each of these techniques is

likely to inherently limit convergence and be useful in extended evolution, although in most of the above cases the primary goal was more effecient use of the test data rather than extending the evolutionary process.

Similarly, Chen et al. (Chapter 13) presented research on co-evolving traders in double auction markets. Their primary goal was to determine the effect of rational traders on the efficiency of double auction markets. However, by incorporating co-evolution they developed a system that is less likely to converge; instead, the evolving traders will continue to adopt novel strategies in response to each others' changes.

However, while co-evolutionary dynamics will delay, or even completely avoid, convergence, it may or may not lead to improved overall results. Instead of improving, the individuals may be caught in a circular arms race – the Red Queen effect.

A more direct approach to avoiding convergence is direct insertion of new genetic material. Thus, for example Korns and Nunez (Chapter 14) and McConaghy et al. (Chapter 8) used Age-Layered Populations (ALPs) as a means to inject new individuals into the evolutionary system.

A number of other papers included techniques that may delay or avoid convergence, although that was not the primary goal and thus their effect on convergence was not measured directly. For example, Moore et al.'s (Chapter 9) overall goal was to determine whether a shift from the relatively simple "artificial evolution" described at the beginning of this chapter to a much more complex "computational evolution" would improve the overall performance of the evolutionary process. The changes they made included allowing the genetic operators themselves to evolve over time, and allowing evolving individuals to build an archive of building blocks for use in future generations. These changes do not produce a dynamic fitness function, which is known to delay convergence, but they do change the shape of the search landscape by changing its interconnectedness. As new operators evolve or new building blocks are added from the archive the inter-connectedness of points in the search space change. The "cultural algorithm" used by Reynolds et al. (Chapter 16) also modifies the scope and behavior of the GP operators. These dynamic changes to the search landscape may delay convergence in the same way that dynamic fitness functions do.

Avoiding Code Bloat in Extended Evolution. In order to extend the evolutionary process indefinitely it is necessary to control code bloat, the tendency of evolving individuals to grow without bounds. Besides leading to large increases in evaluation times and memory use, bloated individuals are difficult or impossible to interpret, leading to opaque candidate solutions with unknown trustworthiness.

Probably the most commonly included technique that would limit bloat was the use of Pareto optimization with improved fitness and limited individual size and/or complexity as the twin goals. Pareto optimization was used by Heywood and McIntyre (Chapter 4), Korns and Nunez (Chapter 14), Kotanchek et al. (Chapter 10), and McConaghy et al. (Chapter 8). However, in many cases the primary rational for including Pareto optimization was not to control bloat for computational efficiency reasons. Instead Pareto optimization was included with the specific goal of generating simpler and hence human readable solutions, avoiding the inclusion of unnecessary variables, or avoiding overfitting.

Kim et al. (Chapter 12) took a more direct approach and constrained the GP process to a very limited set of allowed tree sizes and shapes. Again the primary goal was to generate understandable solutions that avoid overfiting, but a clear side effect is negating the problem of code bloat making extended evolution a possibility. McConaghy et al. (Chapter 8) also put limits on the topologies that could be evolved. The primary goal was to limit the GP search process to (analog circuit) topologies that a human designer would be likely to trust and therefore use. But again this has the side effect of limiting bloat.

Bongard and Lipson's GP system (Bongard and Lipson, 2007) included a *snipping* operator that simplifies and restructures the evolving models. When using snipping, subexpressions in an older model are sometimes replaced with a constant drawn from the range of the subexpression. This reduces overall code size and, as with many other bloat control methods, reduces overfitting.

Several papers also expanded upon our understanding of the causes of bloat. In Chapter 7 Poli et al. characterize the effects of elitism on bloat. The results show that elitism decreases bloat for most of a GP run, although it can slightly increase bloat in the very earliest generations. They show that because elite individuals are copied from earlier generations when the average population size was smaller, their average size is slightly smaller than the average size for the whole population. Thus, their presence lowers the average individual size for the population. This difference is typically fairly small, so bloat is not entirely halted, but higher levels of elitism do limit growth further. Surprisingly they found that elitism up to 30% of the population size could be used without significantly lowering fitness gains.

In Chapter 6 Soule studies the effect of crossover bias on bloat. Crossover bias is the tendency of standard GP crossover to produce a Langrange distribution of tree sizes. This heavily over-samples the smallest, and least fit, trees leading to selection of the few sampled larger trees and rapid tree-size growth. Soule shows that even relatively small changes to the fitness function can significantly change the population distribution, but that it maintains the long tail characteristic of the Langrange distribution, which still favors growth.

Run-time Metrics for Extended Evolution. An important sub-theme within the general theme of extended evolution was how to best monitor the progress of extended evolutionary runs. Clearly, if the proposal is to allow evolution to run indefinitely selecting sample solutions as necessary, it is important to monitor the run to assure that it is still progressing in a useful direction.

In Chapter 2 Worzel et al. use "heat maps" to monitor GP runs (Almal et al., 2007) The results show that heat maps can be quite effective in visualizing diversity and convergence within a population. Significantly the heat maps effectively visualize evolving structures, whereas other run-time metrics of diversity typically return scalar values based on the 'diameter' of the population. This makes it possible to follow a population's evolution through genotype space in addition to phenotype space and makes it much easier to determine if dynamics such as speciation are taking place.

Soule's exploration of the distribution of tree sizes within the population shows that looking at the full distribution, rather than simply averages and modes, can be important to understanding bloat. Researchers using Pareto optimization advocated plotting the full Pareto front at regular intervals as an effective method of tracking the evolutionary process during extended runs.

Reuse

Another significant theme of the workshop was incorporating reuse into GP systems. Reuse allows current populations to (re)use structures developed in previous generations or even previous runs. Crossover is a primitive mechanism for reuse, as it allows subtrees evolved in earlier generations to be copied into new trees. However, it is severely limited by the need to make 'lucky' selection of both the source sub-branch and its destination position. Thus, current research into reuse is aimed at improving the probability that useful building blocks will be conserved and reused effectively.

In the most straightforward approach, the GP system of Moore et al. (Chapter 9) generates an archive of solution building blocks for reuse. Tests with and without the archive show that its use significantly improves performance. McConaghy et al. (Chapter 8) used a more sohpisticated form of reuse that included hierarchical relations. This allowed simpler building blocks to be reused in hierarchically appropriate positions, increasing the probability that they would be used effectively.

Reynold's (Chapter 16) system uses cultural knowledge to direct the generation of new individuals. Although this approach does not constitute reuse in the traditional sense of reusing actual building blocks it clearly (re)uses knowledge gained in previous generations by incorporating that knowledge into the "culture."

Bongard and Lipson's snipping (described above under controlling bloat) may improve reuse by compacting unnecessarily complex subexpressions. Such compacted subexpressions are more likely to be moved through crossover without being broken up, thus making their reuse more likely.

A number of researchers also used grammar based GP, also known as grammatical evolution, e.g. Korns and Nunez, (Chapter 14), and McConaghy et al. (Chapter 8). In grammatical evolution the evolved structures are grammars that define the rules for building the actual executable that is evaluated. Grammars allow very compact defintions of general structural forms. These compact definitions may make reuse more likely, first, because evolution can manipulate general forms rather than specific structures. Whereas the utility of a specific structure is likely limited by its context, making reuse difficult, a general form as defined by a grammatical rule is more likely to be useful in a variety of contexts, making resue more likely. Second, because the rules are both compact and clearly defined it is easier for operators like crossover to manipulate entire rules without disrupting them, making reuse more likely.

Expert Knowledge

The final major theme, both within the papers and during the workshop discussions, was how to use domain specific, expert knowledge to improve performance. Approaches to the use of expert knowledge divided into two broad categories: using expert knowledge in advance to design appropriate structures and operators, and using expert knowledge after a run to assess the results. Many participants advocated both approaches.

McConaghy et al. (Chapter 8) extensively used expert knowledge both before and after each run and, perhaps even more significantly, their system is designed to generate expert knowledge, characterized in traditional formats such as decision trees, for later evaluation and use by human experts. In advance of the runs expert knowledge was use to define the space of possible topologies as a hierarchically composed set of domain-specific building blocks. This both restricted the search space and helped guarantee that the the evolved individuals could be considered trustworthy because they followed established hierarchies. This approach implicitly assumes that an expert will examine the generated solutions. In addition, the system is explicitly designed to automatically generate decision trees that relate performance specifications to topology choice, information on the relative importances of topology and parameters on performance, and whitebox models.

Similarly, Kim et al. (Chapter 12) used domain specific expert knowledge to define the genotype structures (and GP expert knowledge to define the operators) to constrain the GP search space in a financial modeling problem. The primary goal was to evolve relatively simple models that generalized well and avoided

overfitting. In addition, there was an expectation that an expert would examine the evolved models before they were actually used. Thus, expert knowledge was used both in advance to constrain the run and after the run.

Moore and at al. (Chapter 9) use expert knowledge in the form of Tuned ReliefF scores (Tuned ReliefF is a varient of traditional ReliefF scores designed for human genetics problems). Tuned ReliefF estimates the quality of attributes in the data set. Moore and et al.'s GP system then uses these scores in determining the probability of a function being added, deleted, or copied within an existing individual.

For Worzel et al. (Chapter 2) it was explicitly assumed that an expert would examine solutions after the run to determine whether they were biologically reasonable. Many of the practitioners using Pareto optimization to limit the complexity of the evolved structures also assumed, either explicitly or implicitly, that an expert would examine solutions after the run to determine whether they were reasonable to implement.

It is worth noting that this general approach to using GP, evolving solutions that are then vetted by one or more human experts, reflects GP's fairly unique ability to generate innovative solutions. The evolved results are treated very much the way a novel solution by a human engineer would be. (Note that to be human-understandable, the candidate solutions must be of reasonable size and complexity, a key reason to control bloat, as mentioned earlier in this chapter.) Once one designer develops a novel approach other experts are called in to 'vet' the solution before it is used. A similar vetting is not required for most other optimization techniques for the simple reason that they are unlikely to generate solutions which are novel enough to require it.

4. Discussion

An interesting feature of these themes is that they are very tightly coupled. For example, effective use of the test data is important for extended GP, as poor use of the test data is likely to both slow evolution to the point were extended evolution is infeasible and lead to convergence, making extended evolution pointless. Similarly, reuse was independently viewed as an important goal for improving GP performance, but it is likely to be most important in extended GP where there is more time to leverage reuse, e.g., to build hierarchical solutions. Another example of this coupling is domain specific expert knowledge which can be used to improve GP performance directly, but can also be used to design structures and operators that both optimize reuse and increase the effectiveness of extending evolution. Thus, advances targeting one of these themes is likely to generate parallel advances in the others.

This coupling also means that many of the most promising techniques address several themes at once. The broadest example may be Pareto optimization of

performance and complexity. It limits over-fitting on overly complex data sets, limits bloat, and helps generate solutions that can be understood and thus vetted by domain experts.

However, this coupling also has a downside. Because the issues are tightly interconnected and the techniques to address them often impact several critical areas of GP evolution, it can be very difficult to identify the root cause of a particular technique's contribution to performance improvement. E.g. does competitive co-evolution of the training cases produce better performance because it improves the use of the test data or because it limits convergence or are the performance improvements a subtle combination of the two? This is a critical question. In order to reliably apply GP, and to create "turn-key" GP systems for use by non-GP experts, we must know what the root difficulties of a problems are and what techniques most directly and effectively address those problems.

Despite some remaining questions regarding the underlying causes of the improvements, a number of techniques have become fairly widespread, at least among practitioners who deal with industrial scale, real world problems. These commonly adopted techniques include:

- Pareto optimization, particualrly as a means to limit size/complexity;
- Some version of Age Layered Populations (or more generally some form of niching);
- Data partitioning (either a priori or adaptively, e.g. via co-evolution) to limit and/or focus the training set(s);
- Archiving and other forms of reuse;
- Cluster computing;
- Adding noise (if not already present); and
- Ensemble/team solutions.

Currently many practitioners take a "kitchen sink" approach. They keep adding techniques, including many from the preceeding list, until satisfactory results are obtained. This approach is often successful and may be necessary given our still limited knowledge regarding what features make a specific problem difficult and what techniques most effectively address those features. However, it seems unlikely that the kitchen sink approach is the most efficient.

This explains the continued need for frequent communication between the practitioners and the theoreticians. It is usually the practitioners who find the hard problems and devise techniques to overcome those difficulties. But if GP is to continue to advance as craft and then as a science and not remain an art,

we also must be able to identify the factors that make a problem difficult and the techniques that will most efficiently and effectively address those factors, which requires a deeper theoretical understanding of both the problems that arise when using GP and the techniques we use to address those problems.

References

Almal, A. A., MacLean, C. D., and Worzel, W. P. (2007). Program structure-fitness disconnect and its impact on evolution in GP. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 9, pages 145–160. Springer, Ann Arbor.

Banzhaf, Wolfgang, Beslon, Guillaume, Christensen, Steffen, Foster, James A., Képès, François, Lefort, Virginie, Miller, Julian F., Radman, Miroslav, and Ramsden, Jeremy J. (2006). From artificial evolution to computational evolution: a research agenda. In *Nature Reviews: Genetics*, volume 7, page 729—735.

Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA.

Becker, Ying, Fei, Peng, and Lester, Anna M. (2006). Stock selection : An innovative application of genetic programming methodology. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12, pages 315–334. Springer, Ann Arbor.

Bongard, Josh and Lipson, Hod (2007). Automated reverse engineering of nonlinear dynamical systems. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 104, page 9943—9948.

Collet, Pierre and Schoenauer, Marc (2003). GUIDE: Unifying evolutionary engines through a graphical user interface. In Liardet, Pierre, Collet, Pierre, Fonlupt, Cyril, Lutton, Evelyne, and Schoenauer, Marc, editors, *Evolution Artificielle, 6th International Conference*, volume 2936 of *Lecture Notes in Computer Science*, pages 203–215, Marseilles, France. Springer. Revised Selected Papers.

Kantschik, Wolfgang and Banzhaf, Wolfgang (2002). Linear-graph GP—A new GP structure. In Foster, James A., Lutton, Evelyne, Miller, Julian, Ryan, Conor, and Tettamanzi, Andrea G. B., editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 83–92, Kinsale, Ireland. Springer-Verlag.

Korns, Michael F. (2006). Large-scale, time-constrained symbolic regression. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Pro-*

gramming Theory and Practice IV, volume 5 of *Genetic and Evolutionary Computation*, chapter 16, pages –. Springer, Ann Arbor.

Kotanchek, Mark, Smits, Guido, and Vladislavleva, Ekaterina (2006). Pursuing the pareto paradigm tournaments, algorithm variations & ordinal optimization. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12, pages 167–186. Springer, Ann Arbor.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, John R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

Koza, John R., Keane, Martin A., and Streeter, Matthew J. (2003). Evolving inventions. *Scientific American*.

Koza, John R., Streeter, Matthew J., and Keane, Martin A. (2004). The challenge of producing human-competitive results by means of genetic and evolutionary computation. In Menon, Anil, editor, *Frontiers of Evolutionary Computation*, volume 11 of *Genetic Algorithms And Evolutionary Computation Series*, chapter 9, pages 73–99. Kluwer Academic Publishers, Boston, MA, USA.

Langdon, W. B. and Poli, Riccardo (2002). *Foundations of Genetic Programming*. Springer-Verlag.

Lenahan, Jack (2006). The synthesis of evolutionary algorithms and quantum computing. *SIGEVOlution*, 1(3):36–39.

Lohn, Jason D., Hornby, Gregory S., and Linden, Derek S. (2005). Rapid re-evolution of an X-band antenna for NASA's space technology 5 mission. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 5, pages 65–78. Springer, Ann Arbor.

McConaghy, Trent, Palmers, Pieter, Gielen, Georges, and Steyaert, Michiel (2007). Genetic programming with reuse of known designs. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 10, pages 161–186. Springer, Ann Arbor.

Moore, Jason H., Barney, Nate, and White, Bill C. (2007). Solving complex problems in human genetics using genetic programming. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 5, pages 69–86. Springer, Ann Arbor.

Peng, Xiangdong, Goodman, Erik D., and Rosenberg, Ronald C. (2007). Robust design of electronic circuits. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 11, pages 187–202. Springer, Ann Arbor.

Riolo, Rick L. and Worzel, Bill, editors (2004). *Genetic Programming Theory and Practice*, volume 6 of *Genetic Programming*, Ann Arbor, MI, USA. Kluwer.

Robinson, Alan (2001). Genetic programming: Theory, implementation, and the evolution of unconstrained solutions. Division iii thesis, Hampshire College.

Ryan, Conor, Nicolau, Miguel, and O'Neill, Michael (2002). Genetic algorithms using grammatical evolution. In Foster, James A., Lutton, Evelyne, Miller, Julian, Ryan, Conor, and Tettamanzi, Andrea G. B., editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 278–287, Kinsale, Ireland. Springer-Verlag.

Soule, Terence, Riolo, Rick L., and Worzel, Bill (2007). Genetic programming: Theory and practice. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 1, pages 1–12. Springer, Ann Arbor.

Spector, Lee and Robinson, Alan (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

Tan, Xuejun, Bhanu, B., and Lin, Yingqiang (2006). Fingerprint classification based on learned features. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 35(3):287–300.

Wu, Annie S. and Banzhaf, Wolfgang (1998). Introduction to the special issue: Variable-length representation and noncoding segments for evolutionary algorithms. *Evolutionary Computation*, 6(4):iii–vi.

Chapter 2

A POPULATION BASED STUDY OF EVOLUTIONARY DYNAMICS IN GENETIC PROGRAMMING

A.A. Almal[1], C.D. MacLean[1] and W.P Worzel[1]
[1]*Genetics Squared Inc.*

Abstract Understanding the cause-and-effect relationship of the control parameters to the end result can significantly improve the efficiency of any machine learning algorithm. Genetic Programming (GP), even in its simplest form, exhibits complex evolutionary dynamics. This makes the task of determining the relationship of control parameter to the evolved population in a genetic programming system decidedly non-trivial. This chapter describes an investigation into the dynamics of GP. We present several different observations made on the relationship of parameters to evolution in a GP system with the help of a method for visualizing a GP population's fitness, structure and content simultaneously.

Keywords: evolutionary dynamics, visualization, structure, fitness

1. Introduction

The evolutionary dynamics of genetic programming are surprising both in their complexity and in their similarity to natural evolution despite the utter simplicity of the evolutionary mechanisms used compared to those present in natural evolution. This paper draws from work done by the authors in (Almal et al., 2007) regarding a method for visualizing the structure and fitness of all of the individuals in a population. This is done by assigning a point on a circle for each terminal and then, starting from the center of the circle, moving $1/(n+1)$ distance towards the point of the next terminal on the circle. The endpoints arrived at in this way are plotted and the fitness of the individual associated with the endpoint is colored in heat map form where the "heat" of

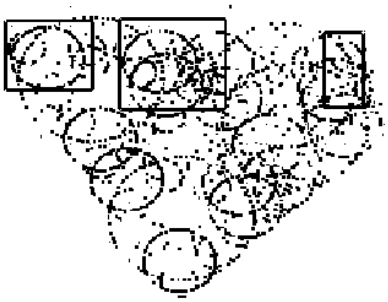


Figure 2-1. Example heat map for a population.

the point plotted is associated with the fitness of the individual. Figure 2-1[1] shows a black and white plot with the boxes highlighting the most fit areas and demonstrating the presence of two distinctly different areas of roughly co-equal fitness. For further details on this technique, the reader is referred to (Almal et al., 2007).

In (Almal et al., 2007) the authors discussed the relationship between genotype and phenotype in GP. We concluded that because the genetic operators of crossover and mutation were genotypic operations, while selection was a phenotypic operation, there was a disconnect between movement through "fitness space" and "structure-content space" that interacted in ways that suggest some areas of structure space to be easier to reach than others, thereby influencing the search path of GP. Moreover, the behaviors observed using this method of visualizing a GP population suggested that GP population behavior was more similar to natural evolution than might be expected.

In this paper, we have visualized the evolution of a number of populations during evolution when the GP control parameters are varied and the GP system is applied to the same problem. Differences between the evolutionary process and outcome are examined and the effect of changing the parameters is discussed.

2. Methods

We used a classification problem from molecular biology as the basis for this study. The input data are normalized gene expression values generated from

[1]Complete sets of color plots are available at http://www.cscs.umich.edu/gptp-workshops/gptp2008 or at http://genetics2.com/Partners%20&%20Collaborators/partners.html

Figure 2-2. Population plot with node frequency displayed proportionally using dot-size to indicate frequency of appearance among a sub-group of the population.

cancer tumor samples. Each value is associated with a single gene's expression in a sample; this sort of study is an association study between gene expression levels and disease outcome. In this case, the study is attempting to find a gene expression signature that can predict the clinical outcome of cancer (Driscoll et al., 2003).

We began by analyzing the data in order to find genes that are most useful in differentiating between patients that have different outcomes. This led to the selection of 10 genes that were used as inputs to the GP system, which then produced predictive classifier rules similar to those described in (MacLean 2004).

One change was made in the method of visualizing the populations created by GP from (Almal et al., 2007): we tracked the number of times particular structures and content combinations were used in a population and, based on this, increased the size of the dot used to express it visually. Figure 2-2 shows an example of this change where the point in the lower right quadrant is very frequently reached at the end of the same structural component, leading to its very large size1. This was done in order to examine the frequency of use of certain subtrees within the population.

It is interesting to note that in the example shown in Figure 2-2, the subtree represented by the large terminal point does not have a particularly high fitness – it is only light green in the usual progression of color coding indicating that while the fitness associated with that subtree must be associated with high-fitness in some cases, it is not always associated with high fitness.

Table 2-1 is a tableau of the GP parameters used in this study. From this starting point, we chose the following parameters and varied them in order to see the effect of the parameters on the evolutionary process: mutation rate, crossover rate, and elitism.

Table 2-1. Tableau of GP settings used for study. Alternative values for mutation and crossover rates and elitism (binary) shown.

Tableau of GP Runs	
Operators	$+, -, *, /, <, >, =, >=, <=, AND, OR, Not, ?$
Population Size	1500
Generations	500
Elitism	1/0
Tournament Size	4
Crossover Rate	0.5/0.9
Mutation Rate	0.5/0.1
Initial Tree Depth	4
Maximum Tree Depth	7
Maximum Terminal Count	15
Number of Demes	16
Migration Frequency (Generations)	10
Migraction Percentage	5%
Fitness Measure	Area Under the Curve (AUC)

The problem being used as a test case is highly multimodal and therefore, due to the relatively high number of inputs, we have restricted the complexity of the predictive functions created by restricting both tree size and terminal count. Our computing system has 16 processors, which process individual demes. Our fitness function is the highest Area Under the Curve (AUC) for the Receiver Operator Characteristic (ROC). Since there are only 96 samples available in this study, there is a real danger of overfitting the data. Because of this we use a n-fold cross-validation approach to test the generality of solutions found across each run (Driscoll et al., 2003).

By using two different very different crossover and mutation settings, we tested the differences in the evolutionary dynamic between these two settings. The higher crossover rate and lower mutation rate tended to converge more quickly on the space it was going to search. An example of this may be seen in Figure 2-3, where the 0.5/0.5 crossover/mutation produced a fairly wide search area at generation 109 while in Figure 2-4, the 0.9/0.1 crossover/mutation rate produced a more tightly focused search area after 109 generations.

Figure 2-3. Crossover/Mutation rate combination 0.5/0.5 produces a diffuse search area after 109 generations.

Figure 2-4. Crossover/mutation rates 0.9/0.1 produces a more concentrated search area after 109 generations.

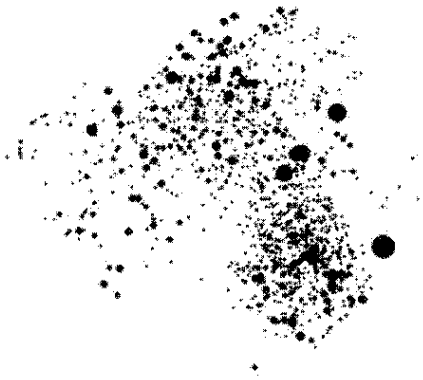


Figure 2-5. Generation 329 from a run with elitism turned off.

Meanwhile the fitness values tend to be lower in the 0.5/0.5 crossover/mutation rate runs after the same amount of time, where the color values tend toward yellow and green compared to colors in the orange-red range for the 0.9/0.1 crossover/mutation rate runs, indicating overall populations of higher fitness.

Turning elitism on and off also produces a notable difference. When elitism is off, it tends to slow convergence noticeably and increases the amount of space searched. This may be seen in Figures 2-5 and Figure 2-6. Figure 2-5 shows a typical plot at generation 499 (the last generation visualized) and Figure 2-6 shows generation 499 with elitism turned on. As may be seen, in Figure 2-5, the distribution of individuals is looser than in Figure 2-6 and there are also several terminals with noticeably larger dots in Figure 2-5. This suggests that more sub-populations exist here than in Figure 2-6 where the single large dot dominates the plot, suggesting that most of the population derives from a single pattern that reuses a very similar subtree repeatedly.

Interestingly, the highest fitness of both populations seem to be very similar, although the individuals in Figure 2-5 with comparatively high fitness are more distributed "geographically" in the plot than the high-fitness individuals in Figure 2-6. In essence, they are well distributed throughout the population in Figure 2-5, whereas in Figure 2-6 they fall within a very tight region.

3. Discussion

In (Almal et al., 2007), the technique of generating heat maps was described and tested on a limited number of runs over a small number of generations. Here these tests were expanded to visually demonstrate different behaviors based on

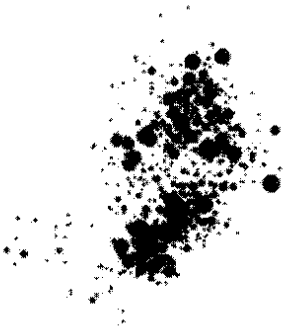


Figure 2-6. Generation 329 from a run with elitism turned on.

the GP control parameter settings. Understanding these behaviors may help to make qualitative judgments as to the correct settings for a problem. For example, if GP tends to overfit a data set, turning off elitism may expand the area searched, potentially finding more neighborhoods of interest and more important subtrees.

While the different settings between mutation and crossover suggest a higher rate of convergence (and more focus on a small area of high fitness) with higher crossover, rates, the causes of this difference are by no means clear. Is it that the higher crossover rate find, and combine building blocks more readily? Or does the higher mutation rate in Figure 2-5 disrupt building blocks as they accumulate? One question that comes to mind is the structural differences of mutational alteration compared to crossover. Since it has been suggested (Angeline, 1997) that crossover is simply a form of macro-mutation, are the differences seen here simply a matter of degree (i.e., would larger scale mutation behave the same way as crossover)?

This work tends to suggest that, though the effects in terms of fitness may be similar, their path through structure-context space may be quite different, possibly because mutation tends to grow the overall tree size less quickly. It would be interesting to apply this approach to artificial problems that were dominated by structural considerations such as the Lid function suggested in (Daida, 2004) or the crossover bias test function developed in (Soule, 2008).

In (Almal et al., 2007), the authors also speculated that speciation events could occur within GP by sudden jumps due to high-tree crossover. While this has not yet been observed, Figure 2-7 shows an early generation that has a very homogenous population. During evolution, this led to a population with

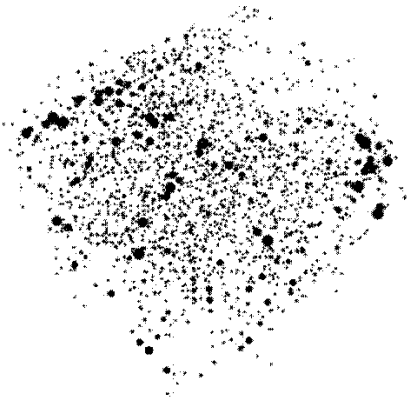


Figure 2-7. Homogenous population at generation 9.

Figure 2-8. Generation 109 with two sub-populations.

two distinct sub-populations (Figure 2-8). This suggests that two "species" can co-exist within a single population within GP.

4. Conclusion and Future Work

The heat maps described here are a useful tool for exploring the dynamics of genetic programming. There are many problems that could be explored with it, including questions regarding the impact of demes on GP, the effect of various

other parameters, the relationship of size limits and population sizes to search dynamics, and the structural limits inherent in different GP systems. Similarly, an exploration of Pareto GP techniques (Smits and Kotanchek, 2004), where tree size and fitness are both used in measuring fitness, would be of value both in understanding how this alters the GP search algorithm and in demonstrating whatever other limits such an approach may bring. Finally, a study of the impact of structure-altering operators as described in (Koza et al., 1996) and (Daida, 2004) would also be of interest.

By adding the "dot sizing" to show the repeated use of structure-contact elements, we can find building blocks and could, over several runs, assemble a list of building blocks for inclusion in assembling a new population. It is interesting to note that while certain key sub-trees recur and have a major influence in the population (such as in Figure 2-2), their fitness is often mediocre at best. This is counterintuitive to the assumption that superior building blocks will inevitably have higher fitness but fits well with the notion that diversity narrows quickly to a limited number of building blocks and then explores how best to combine these building blocks.

On the other hand plots such as those in Figure 2-5 show that such convergence is not a given and that it is dependent on the problem, the parameters used, or both. In short, the genetic programming system is very sensitive both to initial parameters and to specific problems. One-size-fits-all explanations of GP dynamics are probably misplaced.

The combination of this visualization technique with the information theory measures of necessity and sufficiency suggested in (Card and Mohan, 2007) represents an interesting convergence. Since the problems under study here are classification problems, it should be possible to calculate the relationship between different operator sets, the observed evolutionary dynamic, and the measurements suggested in (Card and Mohan, 2007). Conversely, by visualizing a GP system driven by the measurements suggested in (Card and Mohan, 2007), a comparison of the dynamics of such a system could then be explored.

References

Almal, A. A., MacLean, C. D., and Worzel, W. P. (2007). Program structure-fitness disconnect and its impact on evolution in GP. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 9, pages 145–160. Springer, Ann Arbor.

Angeline, Peter J. (1997). Subtree crossover: Building block engine or macro-mutation? In Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L., editors, *Genetic Programming*

1997: Proceedings of the Second Annual Conference, pages 9–17, Stanford University, CA, USA. Morgan Kaufmann.

Card, Stuart W. and Mohan, Chilukuri K. (2007). Information theoretic framework. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 6, pages 87–106. Springer, Ann Arbor. Forthcomming.

Daida, Jason (2004). Considering the roles of structure in problem solving by a computer. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 5, pages 67–86. Springer, Ann Arbor.

Driscoll, Joseph A., Worzel, Bill, and MacLean, Duncan (2003). Classification of gene expression data with genetic programming. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practice*, chapter 3, pages 25–42. Kluwer.

Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin A. (1996). Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 132–149, Stanford University, CA, USA. MIT Press.

Smits, Guido and Kotanchek, Mark (2004). Pareto-front exploitation in symbolic regression. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 17, pages 283–299. Springer, Ann Arbor.

Soule, Terence (2008). Crossover and sampling biases on nearly uniform landscapes. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice VI*, volume 6. Springer, Ann Arbor.

Chapter 3

AN APPLICATION OF INFORMATION THEORETIC SELECTION TO EVOLUTION OF MODELS WITH CONTINUOUS-VALUED INPUTS

Stuart W. Card[1] and Chilukuri K. Mohan[1]
[1]*EECS Department, Syracuse University, Syracuse, NY 13244-4100 USA.*

Abstract Information theoretic functionals have significant benefits as compared with traditional error based indicators of fitness and diversity. Mutual Information (MI), various normalizations of it, and similarity and distance metrics derived from it, can be used advantageously in all phases of Genetic Programming (GP), starting with input selection. However, these functionals are based on Shannon's entropy, which is strictly defined only for discrete random variables, so their application to problems involving continuous valued data requires their generalization and development of robust and efficient algorithms for their calculation. This paper outlines such algorithms and illustrates their application to a noisy continuous valued data set synthesized to test GP symbolic regression systems (Korns, 2007).

Information theoretic *sufficiency* outperforms linear correlation in ranking the relevance of available inputs in this data set. Similar results are obtained on inputs filtered by functions that 'fold' the data, thereby destroying information; ranking these intermediate evolutionary forms, sufficiency again outperforms correlation. Sufficiency also exhibits a distinct threshold separating irrelevant terms from terms that are indeed relevant in regression of these test problems. As a less computationally costly alternative to rankings of entire populations, tournament selection is often used; on this data set, for pairwise tournament selection, sufficiency greatly outperforms correlation. Multi-objective ranking, considering also information theoretic *necessity* to prefer appropriately filtered inputs (over corresponding raw inputs with excess entropies), is foreshadowed.

Keywords: genetic programming, information theory, input selection, building blocks, ensemble models, diversity, fitness, entropy, mutual information, redundancy, synergy, similarity, information distance, evolvability, heritability, sufficiency, necessity, copula

1. Introduction

(Card and Mohan, 2007) presented an information theoretic framework for ensemble modeling by Genetic Programming (GP) and introduced indices that measure fitness and diversity of ensembles without *a priori* knowledge of how multiple constituent models might be composed into a single model. Using those indices for selection, on one discrete and one continuous valued problem, building blocks were found less likely to be lost and more likely to be recombined. Heritability of information should be stronger than that of error, improving evolvability; this was reflected in strong intergenerational correlations (as required by Price's Theorem) on those test problems.

While the application of information theoretic techniques to discrete problems is straightforward, several practical questions have been raised regarding its application to continuous valued data sets, both for input selection when setting up a GP regression problem, and for reproductive and survival selection during subsequent GP runs. To answer these questions, we adopted a continuous valued data set synthesized for testing GP regression systems (Korns, 2007). It is generated by several multiple-input functions applied to pseudo-random inputs. These functions pose severe difficulties for GP or other regression techniques: they depend more strongly upon non-linear than linear interactions among multiple inputs; and they have cyclic dependencies upon the inputs, including both smooth covariations and abrupt behavioral mode switches.

The first question we address is the ability of the information-theoretic approach to distinguish relevant inputs from irrelevant inputs. Hence, unlike (Korns, 2007), which regressed the function outputs against all of, and only, their actual inputs, we also generated additional pseudo-random variables, not actually used as inputs. We calculated various normalized Mutual Information (MI) terms involving the function outputs, actual inputs and spurious 'inputs'. As the scale of these indicators is an artifact of their calculation, affected by noise, computational precision and other factors, we regarded only relative rankings based on those values, not the absolute values themselves.

We calculated how well those input selection rankings performed relative to ideal rankings and compared that against the performance of rankings based on linear correlation. We then applied several single input functions (cos, sin, mod) that 'fold' the data (thereby destroying information) and again ranked expressions. We confirmed that rankings based upon our information theoretic indices outperformed rankings based on correlation.

Section 2 defines information theoretic functionals (both standard and novel) on discrete data. Section 3 outlines a robust procedure for calculating analogs of these functionals on continuous data. Section 4 shows results on test data. Section 5 summarizes findings and ongoing work. Section 6 recognizes those who motivated this work.

2. Preliminaries

For discrete distributions $\mathbf{Y}$ with v distinct values occuring with probabilities $p(\mathbf{y}_i)$, (Shannon, 1948) the *entropy* is[1]

$$H(\mathbf{Y}) = -\sum_{i=1}^{v} p(\mathbf{y}_i)\text{lg}(p(\mathbf{y}_i)) \tag{3.1}$$

Joint entropies of arbitrary numbers of terms are similarly defined. Entropy measures our *a priori* uncertainty about the value of a Random Variable (RV). Entropies can be used to define the *Mutual Information (MI)*

$$I(\mathbf{Y}_k; \mathbf{X}_j) = H(\mathbf{Y}_k) + H(\mathbf{X}_j) - H(\mathbf{Y}_k, \mathbf{X}_j) \tag{3.2}$$

MI measures how much, *on average,* our uncertainty about the value of one RV is reduced by observation of another RV. It can be super- or sub-additive, as reflected by the (Chechik, 2003) *synergy-redundancy index*

$$\frac{I(\mathbf{Y}_k; \mathbf{X}_j, \mathbf{X}_i) - (I(\mathbf{Y}_k; \mathbf{X}_j) + I(\mathbf{Y}_k; \mathbf{X}_i))}{I(\mathbf{Y}_k; \mathbf{X}_j, \mathbf{X}_i)} \tag{3.3}$$

The (Li et al., 2003) *similarity* of two RVs (or *quality* of either as a predictor of the other), and the complementary *normalized information distance*, are

$${}_N I(\mathbf{Y}_k; \mathbf{X}_j) = \frac{I(\mathbf{Y}_k; \mathbf{X}_j)}{H(\mathbf{Y}_k, \mathbf{X}_j)} = 1 - {}_N d(\mathbf{Y}_k, \mathbf{X}_j) = 1 - \frac{H(\mathbf{Y}_k|\mathbf{X}_j) + H(\mathbf{X}_j|\mathbf{Y}_k)}{H(\mathbf{Y}_k, \mathbf{X}_j)} \tag{3.4}$$

In (Card and Mohan, 2007) and previously, for ensemble modeling, the authors decomposed similarity as two distinct objectives. The *sufficiency* of one RV (an input) as an explanation or prediction of another (a target output) is their MI normalized by the target entropy[2]

$$\frac{I(\mathbf{Y}_k; \mathbf{X}_j)}{H(\mathbf{Y}_k)} \tag{3.5}$$

The *necessity* of that variable as a predictor of the target is their MI normalized by the predictor entropy

$$\frac{I(\mathbf{Y}_k; \mathbf{X}_j)}{H(\mathbf{X}_j)} \tag{3.6}$$

[1] We use the notation lg for log_2, ln for log_e and log for log_{10} in equations, pseudocode and text.

[2] When calculating the sufficiency of a *model,* rather than an input, the normalization is not by the entirety of the target entropy, but rather by only that portion of it which could be explained by a deterministic model: the MI between the target output and the entire set of observable inputs upon which the model could draw.

3. Algorithm

With discrete data, Shannon's entropy is well defined. For continuous data, it is not, so analogs have been defined, including 'differential entropy', which retains some of the useful characteristics of discrete entropy, but does not serve our purposes. With discrete data, state space partitions are crisp; with noisy continuous data, they are fuzzy. Algorithms for computing continuous analogs of discrete entropy, mutual information or information dimension typically involve adaptive bin sizes, complicated bookkeeping and heuristics. To address these difficulties, a new algorithm was developed, based upon copulae and sparse arrays.

The *copula* of a set of RVs describes how their joint distribution would look if all their marginal distributions were uniform. The composition of the copula over the marginal distributions reconstructs the full joint distribution. The copula thus captures all the statistical dependencies among the RVs: it reflects all conditional probabilities without regard for prior probabilities. The divergence of the copula from the uniform multivariate distribution measures the mutual dependence of the RVs. Negating the differential entropy of the copula yields the MI (in the case of 2 RVs) or the redundancy (more generally) and the related *information dimension,*which reveals how many independent components would be required to span the given multivariate data set (less than the total number of variables if there are dependencies). As in (Abayomi et al., 2007), given the d dimensional cumulative distribution $\mathbf{F}(\mathbf{x})$, we may calculate its d marginals $\mathbf{F}_i(\mathbf{x}_i)$ and thence the multivariate copula density $\mathbf{dC}(\mathbf{x})$ corresponding to the copula $\mathbf{C}(\mathbf{x})$ by

$$\mathbf{dC}(\mathbf{x}) = \frac{\mathbf{dF}(\mathbf{x})}{\prod_{i=1}^{d} \mathbf{dF}_i(\mathbf{x}_i)} \tag{3.7}$$

The significance of the copula lies in its non-uniformity and the data may have many observable dimensions, so many regions are likely to contain too few points for robust direct estimates of the joint and marginal densities. In (Abayomi et al., 2007), MI is estimated from a parametric copula that approximately models the data. Another approach is suggested by noting that if all the marginals were made uniform, the multivariate copula density could be estimated by the multidimensional histogram.

A straightforward approach is to sort the d dimensional data set d times, once on each variable, and construct the d dimensional co-occurrence matrix with initially atomic bins. However, with N data points, this requires an array with N^d elements. As information theoretic techniques work reliably only on very large data sets, this would require infeasible storage capacity. Fortunately, there are only N non-zero elements, so sparse array techniques can be used, reducing memory requirements from $O(N^d)$ to $O(Nd)$.

Computational cost of the subsequent pyramidal summation of the sparse array is of the same order as that of sorting the data d times to initialize it, $O(Nlg(N)d)$. Note that d must be less than $lg(N)$ or spurious dependencies will be detected,[3] so overall computational cost is $O(Nlg^2(N))$.

The only heuristic involved is deciding when to halt pyramidal summation. The authors chose to stop at the first maximum of the estimated information dimension, the first maximum of its slope, or at a minimum number of bins based upon N and d, whichever is reached first. This compromises between false negative and false positive detections of dependencies in the data.[4]

```
function copula ( data: array [ N, d ] ): sparse array;
zeroize N d dimensional sparse arrays
   dataBinCounts, binFrequencies, binFreqsPrevious;
lgNumberOfBins = ceiling( lg( N ));
numberOfBins = 2   lgNumberOfBins;
for j = 1 to d do {
   sort data on column j;
   replace column j data values with their ordinal ranks };
for j = 1 to N do dataBinCounts[ data[j,1]... data[j,d]] = 1;
while lgNumberOfBins >= minLgNumberOfBins and not done do {
   binFreqsPrevious = binFrequencies;
   binFrequencies = dataBinCounts / N;
   estInfoDim = entropy( binFrequencies ) / lgNumberOfBins;
   done = we just passed 1st max of estInfoDim or its slope;
   lgNumberOfBins--;
   numberOfBins = 2   lgNumberOfBins;
   contract by 1/2 the length of each dimension
      of dataBinCounts by summing merged cells };
if not done
   then { warn "suspect results"; return binFrequencies }
   else return binFreqsPrevious
```

The $entropy[]$ function call refers to a calculation based directly on the definition of the Shannon entropy. That same function is then used on the returned copula, with any dimensions not of interest collapsed by summation, to estimate entropies. MI estimates are normalized by target, model or joint entropy estimates *from the same calculation* to yield sufficiency, necessity or similarity indices to rank alternatives for tournament selection: the values are arbitrarily scaled; only their relative rank order can be used.

[3]This is not a restriction of the algorithm but of the general problem of inferring dependencies from multivariate data.

[4]Implementation code in Wolfram Research's *Mathematica* will be provided upon e-mail request to <cards@ntcnet.com>.

4. Results

From (Korns, 2007), we chose the 5 input versions of the 3 hardest problems: $cyclic$ (here denoted simply y_0), which is difficult due to periodic but mostly smooth functions; $mixed$ (here y_1), which compounds the difficulty with periodic abrupt mode switches; and $ratio$ (here y_2), which further compounds difficulties with division by periodic functions of inputs.

$$\begin{aligned}
y_{0a} &= x_0 \text{sin} x_0 \\
y_{0b} &= x_1 \text{cos} x_0 \\
y_{0c} &= x_2 \text{tan} x_0 \\
y_{0d} &= x_3 \text{sin} x_0 \\
y_{0e} &= x_4 \text{cos} x_0 \\
y_0 &= 14.65 + 14.65y_{0a} - 6.73y_{0b} - 18.35y_{0c} - 40.32y_{0d} - 4.43y_{0e}
\end{aligned} \tag{3.8}$$

$$\begin{aligned}
y_{1a} &= \text{mod (floor } (x_0), 4)) \\
y_{1b} &= 1.57\text{log}|x_0| - 39.34\text{log}|x_1| + 2.13\text{log}|x_2| + 46.59\text{log}|x_3| + 11.54\text{log}|x_4| \\
y_{1c} &= 1.57x_0^2 - 39.34x_1^2 + 2.13x_2^2 + 46.59x_3^2 + 11.54x_4^2 \\
y_{1d} &= 1.57\text{sin}x_0 - 39.34\text{sin}x_1 + 2.13\text{sin}x_2 + 46.59\text{sin}x_3 + 11.54sinx_4 \\
y_{1e} &= 1.57x_0 - 39.34x_1 + 2.13x_2 + 46.59x_3 + 11.54x_4 \\
y_1 &= \text{case } y_{1a} \text{ of } \{0 : y_{1b}, 1 : y_{1c}, 2 : y_{1d}, 3 : y_{1e}\}
\end{aligned} \tag{3.9}$$

$$\begin{aligned}
y_{2a} &= \text{mod (floor } (x_0), 4)) \\
y_{2b} &= \frac{1.57x_0}{39.34x_1} + \frac{39.34x_1}{2.13x_2} + \frac{2.13x_2}{46.59x_3} + \frac{46.59x_3}{11.54x_4} \\
y_{2c} &= 1.57x_0 + 39.34x_1 + 2.13x_2 + 46.59x_3 \\
y_{2d} &= \frac{1.57\text{sin}x_0}{39.34\text{tan}x_1} + \frac{39.34\text{sin}x_1}{2.13\text{tan}x_2} + \frac{2.13\text{sin}x_2}{46.59\text{tan}x_3} + \frac{46.59\text{sin}x_3}{11.54\text{tan}x_4} \\
y_{2e} &= -39.34\text{log}|x_1| + 2.13\text{log}|x_2| + 46.59\text{log}|x_3| + 11.54\text{log}|x_4| \\
y_2 &= \text{case } y_{2a} \text{ of } \{0 : y_{2b}, 1 : y_{2c}, 2 : y_{2d}, 3 : y_{2e}\}
\end{aligned} \tag{3.10}$$

As in (Korns, 2007), the log function is made safe by adding $.000001$ to the absolute value of its input.

Also as in (Korns, 2007), observations of function outputs y_k are corrupted by uniformly distributed gain noise with a default noise weight w of 20% according to

$$y_k(1 + \text{rand}(-w, +w)) \tag{3.11}$$

In our first experiment, 10 RVs x_0 through x_9 were generated, of which the first 5 were used as inputs to each of the 3 functions. In (Korns, 2007), one million points were generated, with each RV uniformly distributed in [-50..+50].

Initially, to limit time and memory requirements, we generated only 2^{16} points; to limit the information loss, in this reduced data set, due to the folding functions invoked (cos, sin, tan, mod, squaring), the range of each RV was restricted to [-12.5..+12.5]. Single inputs were ranked using correlation and sufficiency. Ensembles up through the correct size (5 inputs) and one size too large were ranked using only sufficiency.

In the tables below, the total number of inputs per ensemble in the entropy calculation is m, the target function is f, the relative ranking performance score of correlation is r and the relative ranking performance score of sufficiency matching on n or more of the m inputs in the ensemble is s_n. These scores are the average of the ordinal positions of correct sublists of inputs, subtracted from that average in a random ranking, normalized by that difference in an ideal ranking; that is, a score of 1 is ideal, 0 is random and -1 is exactly backwards.

Table 3-1. Correlation *vs* sufficiency, 2^{16} data points.

m	f	r	s_1
1	y_0	0.56	0.97
1	y_1	0.90	0.94
1	y_2	0.64	1.00

The first experiment was repeated with data sets of 2^{20} (just over one million) points in [-50..+50], limiting time and memory requirements by considering only single inputs and pairs rather than larger ensembles. The larger data set confirmed the results observed with the smaller, and improved average performance of both correlation and sufficiency, except correlation on y_0 ($cyclic$), which it worsened.

Ranking single inputs, compared with linear correlation, information theoretic sufficiency worked slightly better on y_1 ($mixed$), significantly better on y_2 ($ratio$) and dramatically better on y_0 ($cyclic$).

Table 3-2. Correlation *vs* sufficiency, 2^{20} data points.

m	f	r	s_1
1	y_0	0.45	1.00
1	y_1	0.93	1.00
1	y_2	0.85	1.00

Table 3-3. Sufficiency at different ensemble sizes, 2^{16} data points.

m	f	s_1	s_2	s_3	s_4	s_5
1	y_0	0.97	-	-	-	-
1	y_1	0.94	-	-	-	-
1	y_2	1.00	-	-	-	-
2	y_0	0.66	0.83	-	-	-
2	y_1	0.67	0.86	-	-	-
2	y_2	0.69	0.90	-	-	-
3	y_0	0.62	0.79	0.92	-	-
3	y_1	0.61	0.76	0.87	-	-
3	y_2	0.63	0.81	0.93	-	-
4	y_0	0.59	0.73	0.86	0.93	-
4	y_1	0.58	0.70	0.82	0.92	-
4	y_2	0.59	0.72	0.84	0.94	-
5	y_0	0.57	0.66	0.77	0.88	1.00
5	y_1	0.57	0.66	0.77	0.87	0.99
5	y_2	0.57	0.68	0.79	0.90	1.00
6	y_0	0.55	0.63	0.72	0.82	0.99
6	y_1	0.55	0.63	0.72	0.82	0.97
6	y_2	0.55	0.65	0.75	0.95	0.99

Ranking variously sized input ensembles, sufficiency achieved on average more than 90%, and at the correct ensemble size (5) more than 99%, of ideal performance.

The first experiment was repeated 6 times with 2^{16} points and 6 times with 2^{20} points. Little variation was observed between runs, which require considerable time, so there were no further repetitions. Results were averaged to yield the above tables.

In the second experiment, again 10 RVs were generated, of which 5 were used as inputs to each function; but instead of using the same 5 the same way,

Table 3-4. Sufficiency at different ensemble sizes, 2^{20} data points.

m	f	s_1	s_2
1	y_0	1.00	-
1	y_1	1.00	-
1	y_2	1.00	-
2	y_0	0.69	0.88
2	y_1	0.69	0.86
2	y_2	0.69	0.90

different assignments of variables to function inputs were made, as shown in the table below.

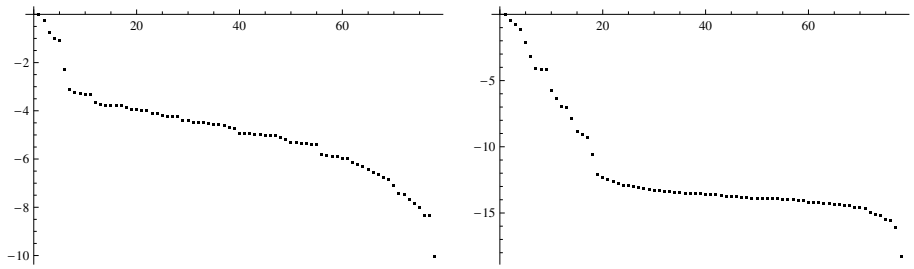
Table 3-5. Assignments of variables to function inputs.

datum	y_0	y_1	y_2
d_{0a}	-	-	-
d_{0b}	-	-	-
d_1	-	-	x_0
d_2	-	x_0	-
d_3	-	x_1	x_1
d_4	x_0	-	-
d_5	x_1	-	x_2
d_6	x_2	x_2	-
d_{7a}	x_3	x_3	x_3
d_{7b}	x_4	x_4	x_4

2 RVs are 'red herrings', not used as inputs to any of the functions; 2 RVs are used as inputs to all the functions; 3 RVs are each used as an input to only a single function; and 3 RVs are each used as an input to all but one of the functions. The data was generated this way to evaluate information theoretic indices as detectors of dependencies between unlabelled observables where input/output relationships are *a priori* unknown.

The plots below are of sorted lists of logarithms of relative strengths of all pairwise linear correlations and information theoretic sufficiencies. They both exhibit distinct noise floors. We consider all strengths above their respective floors to be identified apparent dependencies.

We compare correlation with sufficiency on the basis of their correct and incorrect identifications.



(a) correlations: 10 (b) sufficiencies: 19

Figure 3-1. Sorted log normalized strengths: number above respective noise floor.

On y_1, correlation correctly identified 4 of 5 inputs, missing d_6 (its x_2). On y_2, it had 2 false negatives, d_4 (its x_0) and d_{7b} (x_4); plus it had a false positive, d_3. On y_0, it failed to identify any inputs at all. Correlation also flagged the mutual dependency between y_1 and y_2. It showed one other spurious dependency, between d_{0b} and d_6. Overall it showed 10 dependencies above its noise floor, of which 8 were valid, out of 18 actual dependencies in the data.

Sufficiency correctly identified all 5 inputs of all 3 functions, with no false negatives and no false positives. It flagged all 3 pairwise mutual dependencies of the 3 functions. Sufficiency showed one spurious dependency, between d_2 and d_3; its apparent strength was barely above the noise floor, and less than that of all real dependencies. Overall it showed 19 dependencies above its noise floor, of which 18 were valid, those being all 18 actually present in the data.

In the third experiment, RVs were generated as in the second, and to their set was added cos(d_1), sin(d_1), mod(d_1,4), cos(d_2), sin(d_2), mod(d_2,4), cos(d_4), sin(d_4) and mod(d_4,4). These are functions that 'fold' the inputs, thereby destroying information, applied to the 3 RVs that are each used in one test function only. A different 2 of these 9 terms appear in each of the 3 test functions.

The increased number of dependencies degraded detection of relevant inputs by both correlation and sufficiency; this argues for input selection as a separate pre-processing stage. On y_1, correlation and sufficiency both found dependency on mod(d_2,4), but only correlation on sin(d_2). On y_2, only sufficiency found dependency on mod(d_4,4) and sin(d_4). On y_0, only sufficiency found dependency on cos(d_1), and neither method on sin(d_1). Altogether, correlation detected 2, and sufficiency detected 4, of the 6 dependencies of the test functions on the intermediate terms introduced in the third experiment. Sufficiency also had a false positive, an apparent dependency of sin(d_4) on d_{7a}.

The foregoing experiments required calculation of many-dimensional copulae, which is computationally costly and degrades the sensitivity of the method, but is necessary for commensurate calculations of all the indices, to enable, for instance, fitness proportional selection. A less computationally costly and more

sensitive alternative, that is adequate for tournament selection, is to calculate a lower-dimenional copula for each comparison. For instance, if the strength of a putative dependency of y_1 upon d_{0a} is to be compared against that of y_2 upon d_{7b}, only those variables need be considered, in a 4-dimensional copula.

In the fourth experiment, the 2 false inputs, 8 actual inputs and 3 outputs again were treated as 13 unlabelled observables. All their 78 pairwise potential dependencies were considered. Of the 3003 binary tournaments possible between strengths of putative dependencies, only 1080 were considered: those between false and actual dependencies, treating this as a classification problem; relative strengths of 2 false or 2 actual dependencies were not compared.

On these 1080 classifications, linear correlation was correct 644 times and incorrect 436 times, for an accuracy under 60%, corresponding to a performance score (relative to ideal *vs* random classifications, as in the relative ranking performance scores of the first experiment) under 0.20; whereas information theoretic sufficiency was correct 1059 times and incorrect 21 times, for an accuracy over 98%, corresponding to a performance score over 0.96.

Another way of looking at these results is to consider, for each of the 78 strengths, how many tournaments it would win and how many it would lose against the other 77, and using those statistics to rank them; this is the approximation of fitness proportional by binary tournament selection. Correlation correctly ranks 5 actual dependencies above its first false positive; it ranks the 18th and last actual dependency in position 76 of 78; if the top 1/3 of its rankings were selected, it would have 17 false positives and 9 false negatives. Sufficiency correctly ranks 14 actual dependencies above its first false positive; it ranks the 18th and last actual dependency in a tie of positions 23 through 25 of 78; if the top 1/3 of its rankings were selected, it would have 8 false positives and 0 false negatives.

The four experiments above were reported at GPTP-2008. Participants there, while satisfied that these results had favorably answered the question of applicability of information theoretic techniques to input selection, asked for more evidence of their applicability to fitness evaluation of intermediate forms (neither simple inputs nor final evolved models) during GP runs. Complex subexpressions of the defining equations, that are higher-order building blocks, requiring only a single additional layer of interconnection to correctly complete reconstruction of those equations, may be regarded as 'penultimate forms'. For our test problems, these have been broken out as constituent top-level subexpressions in definitions 3.8, 3.9 and 3.10. A preliminary fifth experiment involving these forms was conducted after the workshop.

In the fifth experiment, for each of the 3 test problems, on 2^{16} data points, an 11-dimensional copula (including all 5 raw inputs, all 5 penultimate forms and the target output), was calculated. For all 3 test problems, the estimated suffiency, necessity and similarity against the target output, of the ensemble of

all 5 penultimate forms, were stronger than the corresponding indices of the ensemble of all 5 raw inputs.

Likewise, in all cases but one, the weakest of the 5 sufficiencies of the penultimate forms was stronger than the strongest of the 5 sufficiencies of the raw inputs. The one exception was on $mixed$, where raw input x_3 appeared to be a slightly stronger predictor of y_1 than did penultimate form y_{1d}. We 'zoomed in' by calculating, on 2^{18} data points, a 3-dimensional copula involving only those 3 variables; this higher resolution measurement correctly ranked the penultimate form above the raw input on all 3 information theoretic indices.

Those familiar with information theory may object that an increase in sufficiency of the ensemble of all 5 penultimate forms over that of the ensemble of all 5 raw inputs violates the Data Processing Inequality. In response we can only remind the reader that Shannon's entropy is defined only for discrete data, and that our estimated analog of it for continuous data is sensitive to the locations and widths of data bins relative to topological foldings of the data. This is harmful, in that intuitions learned using information theoretic techniques on discrete data cannot always be relied upon when working with continuous data; but it is also helpful, in that index values improve as topological foldings are discovered by the GP and reflected in the evolving genotypes.

5. Conclusions

Information theoretic *sufficiency* of continuous valued data can indeed be calculated robustly and efficiently. On this test data, it outperformed linear correlation in ranking single inputs, approached ideal performance in ranking input ensembles, and exhibited a distinct and meaningful noise floor; thus it is an appropriate pre-processor for GP input selection. An extended theoretical treatment of this application, especially measurement of input epistasis as information theoretic synergy, may be found in (Seo and Moon, 2007).

The superior performance of sufficiency (*vs* correlation) on pairwise comparisons satisfies our expectations; but the more important (we believe, unique) benefit is that sufficiency and other information theoretic indices can be calculated for arbitrarily sized ensembles of inputs and outputs, *without assuming any particular method of combination.*

Sufficiency also correctly identified (albeit with a couple of false negatives and a single false positive) simple expressions (derived from relevant inputs by filtering with single input, information destroying, 'folding' functions) that are basic building blocks for starting to regress the test problems; thus it is also an appropriate fitness function early in GP runs. In preliminary experiments, it also correctly ranked 'penultimate forms' above raw inputs, and ensembles of penultimate forms above ensembles of raw inputs; thus it also appears promising as a term for a multi-objective GP fitness function later in GP runs.

Another appropriate objective term is information theoretic *necessity.* In preliminary experiments with continuous data, and in other experiments with discrete data (to be reported in the first author's forthcoming dissertation), necessity further distinguished appropriately filtered inputs and complex sub-expressions from raw inputs and ensembles thereof. The theoretical basis for believing that necessity is a pertinent objective is that excess model output entropy contributes to error in the same fashion as residual target entropy, so GP individuals should be selectively rewarded for filtering out such unnecessary and harmful entropy.

In (Card and Mohan, 2007), the authors conjectured that one source of GP hardness (poor evolvability) is weak correlation (poor heritability) between the phenotypic information distance based on Shannon's entropy, and a corresponding genotypic information distance based on Kolmogorov complexity. Independent of this as yet unverified theoretical conjecture, for practical ensemble modeling by GP, these and previously reported results demonstrate the advantages of information theoretic indicators of diversity and fitness for reproductive (epecially non-random mating) and survival selection.

6. Acknowledgements

The authors thank the participants in the Genetic Programming in Theory and Practice 2007 workshop for the discussions that led to this work, especially Martin Keane for asking pointed questions and Michael Korns for supplying the test function data set.

References

Abayomi, Kobi Ako, Lall, Upmanu, and de la Pena, Victor H. (2007). Copula based independent component analysis. Working paper, Duke University and Unknown and Columbia University. http://ssrn.com/abstract=1028822.

Card, Stuart W. and Mohan, Chilukuri K. (2007). Information theoretic framework. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 6, pages 87–106. Springer, Ann Arbor. Forthcomming.

Chechik, G. (2003). *An Information Theoretic Approach to the Study of Auditory Coding*. PhD thesis, Hebrew University.

Korns, Michael F. (2007). Large-scale, time-constrained symbolic regression-classification. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 4, pages 53–68. Springer, Ann Arbor.

Li, Ming, Chen, Xin, Li, Xin, Ma, Bin, and Vitanyi, Paul (2003). The similarity metric. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms*, pages 863–872.

Seo, Dong-Il and Moon, Byung-Ro (2007). An information-theoretic analysis on the interactions of variables in combinatorial optimization problems. *Evolutionary Computation*, 15(2).

Shannon, Claude (1948). A mathematical theory of communication. *The Bell systems Technical Journal*, 27.

Chapter 4

PARETO COOPERATIVE-COMPETITIVE GENETIC PROGRAMMING: A CLASSIFICATION BENCHMARKING STUDY

Andrew R. McIntyre[1] and Malcolm I. Heywood[1]

[1] *Faculty of Computer Science, Dalhousie University. 6050 University Ave. NS. B3H 1W5 Canada*

Abstract A model for problem decomposition in Genetic Programming based classification is proposed consisting of four basic components: competitive coevolution, local Gaussian wrapper operators, evolutionary multi-objective (EMO) fitness evaluation, and an explicitly cooperative objective. The framework specifically emphasizes the relations between different components of the model. Thus, both the local wrapper operator and cooperative, objective components work together to establish exemplar subsets against which performance is evaluated and the decomposition of the problem domain is achieved. Moreover, the cost of estimating fitness over multiple objectives is mitigated by the ability to associate specific subsets of exemplars with each classifier. The competitive coevolutionary model, in combination with a balanced sampling heuristic guides the identification of relevant exemplars and retention of the best classifiers; whereas the EMO component drives the application of selection and variation operators to maintain the diversity of the classifier population. Empirical evaluation is conducted over twelve data sets representative of different application domains, class distributions and feature counts. A benchmarking methodology is developed in order to provide comparison between the proposed model and the deterministic classifier paradigm in general, and the SVM method in particular. Moreover, by adopting performance metrics for classification, model, complexity and training scalability, we are able to provide a more informative assessment of the relative merits of each model.

Keywords: competitive coevolution, cooperative objective, evolutionary multi-objective optimization, problem decomposition, multi-class classification, sub-sampling, wrapper function design, unbalanced data sets.

1. Introduction

Classification represents one of the most widely studied domains of machine learning. The current state-of-the-art is generally considered to take the form of the Support Vector Machine (SVM), where such models are both very accurate and effective across large data sets or those having unbalanced class representations (Japkowicz and Stephen, 2002). Conversely, canonical Genetic Programming (GP) does not scale well as the number of training exemplars or classes increases, and has no implicit resilience to unbalanced representation of exemplar classes. Naturally, the class imbalance issue may be addressed by assuming problem dependent cost functions, whereas various heuristics have been proposed to address multi-class classification and large data sets (including hardware specific solutions). However, the only implicit advantage that canonical GP is able to maintain relative to the generic SVM model is a bias towards indexing subsets of features, thus providing the potential for more transparent solutions than the 'black box' generally associated with SVM models.

On the other hand, one of the strengths of the GP paradigm is the freedom with which the basic machine learning design questions – cost function, representation, and credit assignment – may be addressed. In this work, we are interested in using this freedom to provide a framework for encouraging GP to actually decompose the problem domain as part of the evolutionary cycle. Thus, solutions might take the form of multiple intra-class mappings per class, each individual associated with the solution responding to a non-overlapping subset of class consistent exemplars. Individuals are therefore explicitly rewarded for correct but also unique behaviors. Moreover, the resulting model should also be scalable. That is to say, tens or hundreds of thousands of training exemplars should not result in recourse to hardware specific solutions. The resulting framework is based on a canonical GP kernel, in this case Grammatical Evolution (O'Neill and Ryan, 2001), but makes extensive use of Pareto coevolution and a mechanism for evolving the properties of a 'local' Gaussian wrapper operator. Key characteristics of the resulting Cooperative Multi-objective GE framework – hereafter denoted CMGE – are summarized in Section 2.

In order to evaluate the proposed CMGE framework, we are interested in performing an assessment under multiple criteria: classification, model simplicity and scalability, Section 3. By doing so we demonstrate that the CMGE framework is able to provide competitive classification performance, while also returning parsimonious solutions and scaling gracefully to large data sets, Section 4. Overall conclusions in which some opportunities for future work are identified in Section 5.

2. A Coevolutionary Multi-objective Framework for Problem Decomposition

In order to develop the proposed approach to GP classification we will introduce the framework in terms of the high level pseudo code listing provided in Figure 4-1. This establishes the proposed framework in terms of a five step algorithm having two basic components: cooperative, coevolution for fitness evaluation and problem decomposition (step 2(c) and 2(e), Figure 4-1) and a competitive coevolution archiving for training exemplar subset selection and model building (step 2(d), Figure 4-1).

An interaction between a GP individual and a single exemplar is defined by executing the GP program using the exemplar features as arguments. The result of executing a single program on an exemplar (the interaction) is a real-valued output; mapping the exemplar (from a typically multi-dimensional feature domain) to a one dimensional number line that we refer to as the 'GP Output' space, or simply *gpOut*. Naturally, the mapping process is repeated to map n exemplars to n points on *gpOut*. Such a mapping alone naturally conveys no class information. To do so, canonical GP employs a global membership function (wrapper operator) to impart class labels on the distribution of points in *gpOut*. Such an approach effectively casts all points on the *gpOut* axis that are greater than (less than) zero as in-class (out-class) exemplars, and weights the number of misclassifications by way of a cost function. Such a model implicitly assumes that partitioning the class labels about the origin of the *gpOut* axis is appropriate for the problem domain in question. Under the CMGE model, label assignment is realized by means of a local membership function (LMF) such that class consistent subsets of exemplars should be mapped to the same local neighborhood on the *gpOut* axis.

The basic features of the Competitive Multi-objective Grammatical Evolution (CMGE) classifier are now summarized as follows relative to the pseudo code listing provided in Figure 4-1:

Feature 1: A class-wise random balanced sampling heuristic is enforced such that each class has equal representation in the point population, step 2(a). This policy is assumed as individuals (in the point population) represent exemplar indexes. Such a representation means that there is no structure on which to build search operators, whereas there is evidence that a balanced sampling heuristic provides a bias towards optimizing the AUC performance metric (Weiss and Provost, 2003).

Feature 2: The local membership function is derived from the distribution of points on the *gpOut* axis, step 2(c).i to iv. As such, no attempt is made to incorporate the concept of class labels when deriving the characterization of the individual's local membership function. Instead, we assume that the local membership function is expressed by a Gaussian, the parameters of which are

1. Initialize Learner Population (*LP*);
2. WHILE ! (Stop criteria)
   (a) Point Population (*PP*) := random balance sample of training partition;
   (b) Training Subset (*TS*) := *PP* concatenated with Point Archive contents (*PA*);
   (c) FOR j := 1 to sizeof(*C*)
      i Establish phenotype of individual $I[j]$;
      ii Map *TS* to 1-*d* number line '*gpOut*' of $I[j]$;
      iii Cluster *gpOut* of $I[j]$;
      iv Parameterize Gaussian Local Membership Function (LMF) of $I[j]$;
      v Evaluate $I[j]$ with respect to:
      SSE, Overlap wrt. Learner Archive (*LA*), Parsimony.
      vi Rank $I[j]$ with respect to *LP* and assign fitness;
      vii Replacement (insert $I[j]$ into *LP*);
   (d) Archive *PP*, *LP* members based on outcomes (according to IPCA)
      i Points in *PP* enter *PA* if they provide a distinction;
      ii Learners in *LP* enter *LA* if they ar non-dominated wrt. *LA*;
   (e) Evaluate Stop Criteria (method of consecutive Rank Histogram assessment);

Figure 4-1. CMGE Algorithm. *TS*, training subset; *LP*, learner pop.; *PP*, point pop.; *LA*, learner, archive; *PA* point archive; $I[j]$, classifier 'j'.

derived by first applying a clustering routine to the individual's point distribution on the *gpOut* axis. Having identified the subset of points associated with the most dense cluster, the mean and variance of the Gaussian local membership function for that particular individual are established. Only the subset of points within the cluster as evaluated on a specific individual's *gpOut* axis are associated with the local membership function. This is the first property by which we establish problem decomposition.

Feature 3: At this point we have a set of individuals with their corresponding local membership functions and therefore possibly unique subsets of exemplars established. The class label associated with the individual is established by assuming that the individual takes the label of the exemplar with maximum membership of the Gaussian. With the introduction of class labels we may now characterize fitness over multiple objectives, albeit for the subset of exemplars actually mapped to the Gaussian alone, step 2(c).v. Moreover, the multi-objective GP view provides the opportunity to reward non-overlapping behaviors as well as error minimization; the second property by which problem decomposition is encouraged. However, there is a 'trick' to this process: the estimation of overlap is performed relative to the contents of the learner archive, as established by competitive coevolution, Feature 4. Naturally, having established the relative fitness of individuals, selection and reproduction takes place under a Pareto multi-objective model (Kumar and Rockett, 2002) which encourages diversity without recourse to distance based metrics, step 2(c).vi to vii. This completes the explicitly cooperative aspect of the framework.

Feature 4: Competitive coevolution is now used to identify the best points and classifiers to retain outside of the point and learner populations, step 2(d). To do so, an outcome vector is constructed over the current contents of the point archive and population, relative to the current contents of the learner archive and population. As such this process follows the IPCA algorithm of de Jong (de Jong, 2004), however, any form of this class of competitive coevolution would be appropriate. Special attention is necessary to the derivation of an appropriate mechanism for establishing the outcome vector. In particular this is a real-valued pairwise matrix of the behavior of each individual relative to points. Only individuals that are non-dominated in the Pareto sense (with respect to outcome vectors) represent candidates for archiving. Similarly, only the points making the distinction between dominated and non-dominated learners represent candidates for the point archive. It is the contents of the learner archive that represents the set of individuals comprising the solution.

Feature 5: Stop criteria is established in a problem independent manner by making use of the concept of Pareto rank histograms, step 2(e), as established in the Pareto multi-objective technique adopted in Feature 2 above. Unlike the original GA context in which this concept was derived (Kumar and Rockett, 2002), we also deploy it in a class wise manner. This enables us to declare

classes converged class-by-class, thus providing the ability to redeploy the individuals associated with that class, such that they are reassigned to classes as yet not converged.

Further algorithmic details are available in (McIntyre and Heywood, 2007) and (McIntyre, 2007).

3. Evaluation

Empirical evaluation is generally considered the basis for establishing the significance of different machine learning algorithms. As such the typical approach takes the form of a 'bake-off' between the proposed model and a 'straw man' alternative, as evaluated on a handful of *a priori* selected data sets. When one algorithm is able to establish significance of a one sided statistical test on more of the data sets than the other, a winner is declared. Such an approach is increasingly being questioned (Drummond, 2006). Under such a context, we consider factors central to establishing a better basis for evaluation of GP methods relative to classical deterministic machine learning methods as follows:

Performance measure: Depending on the characteristics of the underlying data set, assuming a single performance metric may lead to very biased results. The poor performance of 'error' or 'accuracy' based metrics has long been recognized (Provost et al., 1998), yet such metrics continue to be employed as the sole basis for comparison. Moreover, this has also lead to ignoring other performance factors such as model, complexity, in favor of metrics purely associated with classification performance. Unfortunately this can lead to 'black box' solutions in which the complexity of the resulting model is ignored, leading to a low acceptance rate of machine learning based solutions.

Inappropriate use of statistical tests: Aside from the relevance of null hypothesis based statistical testing as a whole (Drummond, 2006), a machine learning practitioner is frequently left with an implicit experimental design problem. The non-deterministic methodology of GP requires that evaluation be conducted over multiple runs per data partition. Conversely, deterministic machine learning algorithms only require one evaluation per partition (for a given selection of learning parameters). The evaluation design problem of interest here therefore comes down to establishing how the single performance point from the deterministic algorithm may be compared to multiple points from the evolutionary method.

Data sets: Benchmarking against well known data sets may provide several advantages, including the development of a body of knowledge regarding the attainable level of performance associated with each data set, the implicit biases inherent in the data, and how well particular learning algorithms perform. Conversely, focusing on the same set of data sets also tends to result in learning

algorithms 'over fitting' the characteristics specific to the subset of popular data sets.

In this work we are interested in assessing how the proposed CMGE framework compares to an SVM classifier. As such this could result in the classical 'bake off.' In order to avoid this we attempt to address the above generic problems with the bake off approach, at least in part, by adopting the following measures:

Multifaceted performance evaluation: Performance will be considered from three perspectives: Classification performance, model, complexity and CPU training time. In the case of specific *classification performance* metrics, we are again faced with the inherent problem of choosing a metric that provides an effective summarization for both binary and multi-class domains and unbalanced class distributions (Japkowicz, 2006). One approach might be to utilize Receiver Operating Characteristic curves. However, this also implies adopting a suitable heuristic for defining performance thresholds over a set of models (c.f. CMGE intra-class solutions), as well as being limited to pairwise comparisons (thus, not scaling gracefully under multi-class domains). In this work we will therefore build a 'score' metric that measures the class-wise detection rate under an equal class significance weighting assumption:

$$score = \frac{\sum_{i \in C} DR_i}{\mid C \mid} \tag{4.1}$$

where DR_i is the detection rate of class 'i'; and $\mid C \mid$ is the number of classes.

Model complexity will be assessed through a count of the number of 'elements' used to construct the model; where the elements are naturally a function of the representation assumed by each machine learning model. In the case of CMGE 'elements' are considered synonymous with a normalized instruction count over all individuals participating in a solution, whereas in the case of the SVM paradigm we count the number of support vectors. The CMGE normalization reflects the implicit bias of support vectors to index all features, whereas GP instructions may only index a maximum of two features. This implies that, given a two argument instruction, there are $F-1$ instructions required to index all features, where 'F' is the feature count. The CMGE element count therefore takes the form:

$$CMGEcomplexity = \frac{NumIndividuals \times NumInstructions}{F-1} \tag{4.2}$$

Naturally, equivalence is not implied, as it is in the hands of the practitioner to decide how transparent the ensuing solution might be.

CPU training time will be assessed in terms of a common computational platform (dual core iMac with 2GB RAM, Tiger OS) for four different training

data sets, corresponding to thousands, tens of thousands, 150 thousand and 200 thousand exemplars respectively. As such this is designed to provide feedback in terms of the trade off between fixed memory footprint of CMGE – at the possible expense of accuracy – versus the increasing memory footprint of the SVM implementation.

Normalize relative to deterministic performance point: Models of machine learning such as an SVM, C4.5, or Naive Bayes apply a deterministic procedure for building a model; whereas evolutionary methods may make different design choices on account of their stochastic policy of credit assignment. This results in an inherent disjunction in the number of performance points available for comparison. For each training partition we have one performance point care of the deterministic model, versus fifty or so points under an evolutionary method; performing cross validation does not change this. To this end we will use the performance point from the deterministic model to normalize the distribution of points provided by the evolutionary model. Plotting the result as a box plot establishes the likelihood of the evolutionary method performing any better than the deterministic. Thus, by retaining the distribution of points we are able to directly resolve the significance or cost of assuming a stochastic model relative to the *a priori* post-training performance metric. Naturally, the deterministic model will be evaluated under multiple parameterizations of learning parameters and the best case selected with respect to the same performance function under the training partition.

Seek out a 'diverse' cross section of data sets: Multiple sources for benchmarking data sets are now available e.g., UCI and KDD. Moreover, the major sources are receiving additional data sets on a continuous basis. As such this provides the opportunity to sample data sets with properties such as: different application domains, different exemplar distributions, multiple classes, or feature counts. In addition, various benchmark studies have identified data sets that are in some way 'difficult' to classify (Lim et al., 2000). In this case a total of twelve data sets are utilized, Table 4-1, with the goal of retaining some 'old favorites' with varying degrees of difficulty (e.g., Breast, Iris, Liver, Pima, Wine) as well as introducing large multi-class / multi-feature / very unbalanced data sets (e.g., Census-Income, Contraceptive, ImgSeg, KDD Cup 99, Shuttle, Thyroid).

Parameterization

CMGE. The CMGE model requires parameterization of the basic Grammatical Evolution (GE) model, archive and population sizes, cluster algorithm parameters, and definitions for early stopping, Table 4-2. GE parameters set limits on the maximum length of an individual before and after codon transcription (O'Neill and Ryan, 2001). Variation operators take two basic forms:

Table 4-1. Data Set Characterization. $\mid T \mid$ is the total number of exemplars in the data set; Train (Test) indicate the exemplar count for training (test) sets, or whether a stratified '10-fold' is deployed; 'F' denotes the number of features; and $\mid C \mid$ the number of classes; '†' denotes a class with a representation at less than 0.7% of the training partition.

Dataset	$\mid T \mid$	Train	Test	F	$\mid C \mid$	% Distribution
Boston	506	10-fold	10-fold	13	3	$\approx$ balanced
Breast	675	10-fold	10-fold	10	2	(65):(45)
Census	295 173	196 294	98 879	41	2	(94):(6)
Contra	1 425	10-fold	10-fold	9	3	(43):(22):(35)
Image	2 310	210	2 086	19	7	$\approx$ balanced
Iris	147	10-fold	10-fold	4	3	$\approx$ balanced
KDD99	222 871	145584	77 287	41	5	(60):†:(37):(1.5):†
Liver	341	10-fold	10-fold	6	2	(42):(58)
Pima	768	10-fold	10-fold	8	2	65:35
Shuttle	58 000	43 500	14 500	9	7	(78):†:†:(15.5): (5.65):†:†
Thyroid	7 129	3709	3420	21	3	(2):(5):(93)
Wine	178	10-fold	10-fold	13	3	(33):(40):(27)

canonical and structure preserving. Canonical crossover and mutation tend to be global in their behavior, on account of the action of the grammar, and are therefore applied with a comparatively low likelihood, Table 4-2. Structure preserving variation operators make use of context information returned by genes associated with building terminals during application of the grammar (Harper and Blair, 2005). As such this provides the basis for local variants of crossover and mutation, designed such that the impact is more likely to be restricted to single 'subtrees,' and are therefore applied at a higher frequency. The context free grammar itself is designed to support the four arithmetic operators, sine and cosine, exponential and natural log, square root and index the data set features.

Point and learner archives enforce a fixed upper bound on the memory mechanism used to retain points supporting the Pareto front of Learners identified during competitive coevolution. Separate point and learner archives are retained for each class, whereas a single population is maintained for establishing search diversity, as per the original IPCA algorithm (de Jong, 2004). A constant archive and population limit is maintained for all cases other than the learner population, which is larger to encourage greater learner diversity. The early stopping parameters are modeled on a simplified version of the Pareto histogram method (Kumar and Rockett, 2002) and effectively declare the degree of similarity between sequential generations of the EMO inner loop of CMGE.

Table 4-2. CMGE Parameters. 'Archive' sizes are quoted per class.

Clustering	value	Archive or Pop.	value
α	150	Learner Pop. Size	50
β	155	Learner Archive Size	30
γ_{lower}	0.5	Point Pop. Size	30
γ_{upper}	0.75	Point Archive Size	30
GE	value	Early Stopping	value
max. prog. length	4,096	Min. Difference	0.1
max. Codon count	256	Min. Pop.	20
max. Epochs	500	Faction Evolved	$\frac{1}{5}$
Crossover and Mutation rates: canonical, (structural)			
P(crossover)	0.5, (0.9)	P(mutate)	0.01, (0.9)

Learners and points from other classes continue to evolve, and search resources associated with the converged class are reassigned to the remaining classes. The final set of parameters from Table 4-2 define the radii (α and β) and stop conditions (γ) for the Potential function clustering algorithm (Chiu, 1994) used in CMGE to independently configure the local membership function of each learner.

All of the above remain fixed across all runs, irrespective of data set. Needless to say, no claims are made regarding the optimality of the parameter selection or grammar. Relative to the training partitions of Table 4-1, fifty runs are performed in each case, thus data sets defined in terms of 10-fold cross validation imply a total of 500 runs under CMGE.

SVM. The method of Support Vector Machines (SVM) is based on a quadratic optimization problem designed about a 'kernel function' that maps the original input space to an orthogonal space, the support vectors. Much of the research for providing improved SVM approaches is directed towards establishing more efficient methods for performing the process of optimization. One of the first breakthroughs in this respect was to consider the process as a series of decompositions in which the Sequential Minimal Optimization method provided a very efficient paradigm (Platt, 1998). More recent refinements have included the use of second order information to speed the rate of convergence, while also addressing the additional computational cost of doing so (Fan et al., 2005). The LIBSVM implementation (release 2.85) supports the use of the aforementioned second order model (Chang and Lin, 2007) (see also Chapter 1 in (Bottou et al., 2007)) and is therefore adopted in this benchmarking study.

Unlike the GP scenario, sources of variation in the model are now limited to the SVM learning parameters, of which the principal parameters are the kernel function and regularization parameter C. Two of the most widely utilized kernel functions take the form of the Gaussian (or radial basis) and Sigmoidal function, hence the following evaluation will consider both cases. Three values for the C parameter will be employed $\{1, 10, 100\}$ under each data set, with the SVM model selected corresponding to the best post training performance as evaluated by the 'score' metric of Equation 4.1. In addition the input features will be normalized to the unit interval relative to the default application values (no such normalization is applied under the evolutionary model); the SVM output will take real valued responses, where this represents a significant improvement over the default of binary values for multi-class domains.

One significant difference between SVM and CMGE methodologies with regards to computational resource is the requirement for extensive cache memory support with the SVM paradigm. Under the data sets previously benchmarked by (Fan et al., 2005), two cache memory limits were considered: 40 MB and 100KB, when the largest data set considered consisted of around 15,000 exemplars. In this study the Census data set consists of around 200,000 training exemplars. The resulting SVM memory utility went up to 2 GB; whereas the CMGE model has a constant memory utility, as defined by the union of point population and class archives. The impact of this from a computational perspective will be considered in the following performance evaluation.

4. Performance Evaluation

Following the above discussion, for each kernel function, three SVM models are trained per partition (reflecting different values for C) and then the score metric of Equation 4.1 employed to select the representative solution. Performance of that model is then evaluated on the test partition, again in terms of the score metric. This establishes either a single performance point, or ten performance points, as in the case of the cross-validation data sets i.e., those without an *a priori* defined training partition, Table 4-1. In the latter case we therefore establish a single SVM performance point by assuming the median of the ten test results. The single SVM performance point is then used to normalize the distribution of CMGE solutions. A CMGE distribution is established by taking the fifty best solutions as defined by the score metric over the training partition, post training, and evaluating the model under test. Naturally, data sets with a single training partition will utilize all fifty initializations, whereas the runs from the seven 10-fold cross validation scenarios will assume a sample from the 500 CMGE solutions.

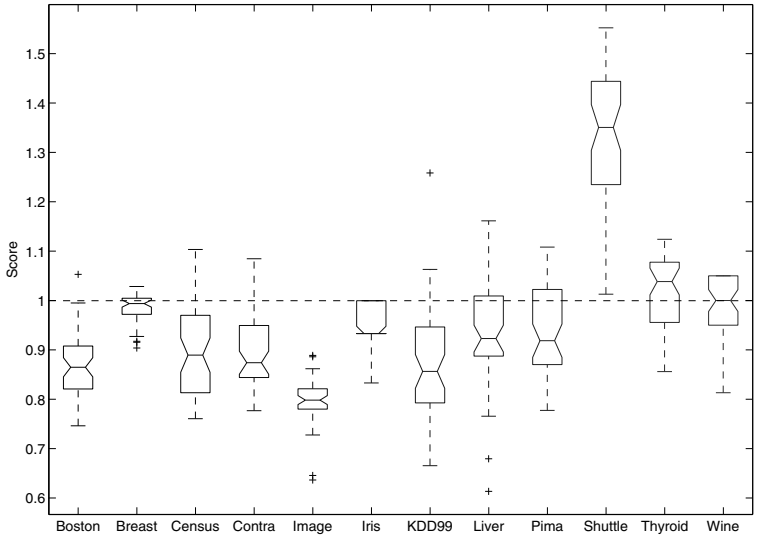


Figure 4-2. Gaussian kernel SVM performance point normalized CMGE score on test. CMGE distribution is summarized in terms of rank statistics of 1st, 2nd (median), and 3rd quartile, with wiskers indicating the limits of the distribution after which outliers are identified. Notches imply the 95th percentile associated with the variance of the median.

Classification Performance

Figures 4-2 and 4-3 detail the distribution of SVM normalized CMGE test partition performance on each of the twelve data sets with respect to Gaussian and Sigmoid kernels respectively. Naturally, results above unity imply that the SVM performance point was bettered, whereas values below unity imply an SVM score bettering the CMGE distribution. Moreover, CMGE distributions are summarized in terms of a box plot, thus we are able to establish the contribution of any outlier behavior and explicitly identify the quartile performance points of each data set.

It is clear that of the two SVM models, the Gaussian kernel results in stronger scores than the Sigmoid on all but the Shuttle data set. Generally, the CMGE model is as capable as the best of the two SVM models for Breast, Iris, Shuttle, Thyroid and Wine data sets. In the case of the Liver and Puma data sets a minimum of twenty five percent of the CMGE results might be expected to perform better. However, the SVM is clearly much stronger under the Boston, Image Segmentation and KDD-99 data sets; and depending on the kernel Census and Contraceptive.

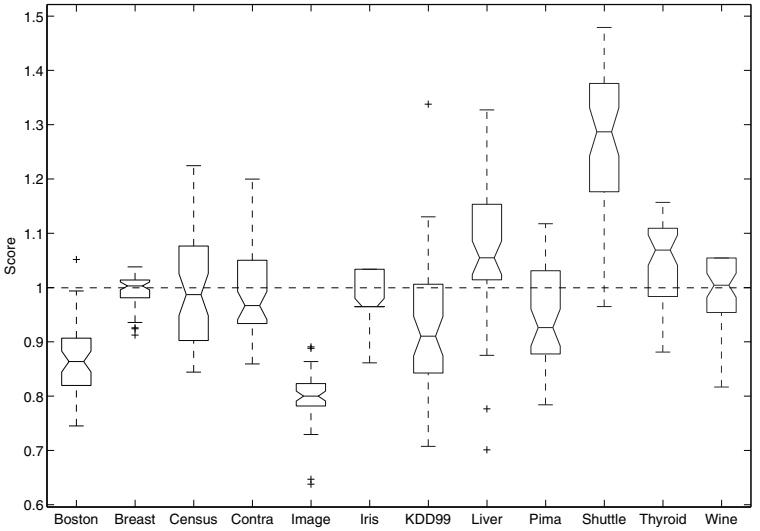


Figure 4-3. Sigmoid kernel SVM performance point normalized CMGE score on test. For comments on the Box Plot, see previous figure.

Table 4-3 details the numerical score for each of the twelve data sets; as such we can discern the specific data sets on which the SVM and CMGE models are most likely missing entire classes, as score values tend towards a multiple of $\mid C \mid$. In the case of both models, a tendency to ignore all but the major class exists on Census, and the two major classes on KDD99 (plus half of the third larger class). On Shuttle, the SVM tends to classify the four larger classes and half of a fifth; whereas the CMGE model focuses on two of the three in Boston and five of seven under Image Segmentation. In short, both CMGE and SVM models have specific strengths and weaknesses, or no free lunch.

Model Complexity

In the case of model, complexity, we again use the SVM performance point as the normalizing factor. As per the discussion in the previous section, an SVM element is considered a support vector, whereas an element under the CMGE model will be considered to be a normalized instruction count taken across all individuals comprising a solution, Equation 4.2. Figure 4-4 summarizes the resulting SVM normalized median complexity of CMGE solutions on each data set with respect to Gaussian and Sigmoid kernel functions. A log scale was necessary on account of the resulting wide variation in solution complexities.

Table 4-3. Median SVM score per data set. Corresponds to the normalization factors used in Figures 4-2 and 4-3.

Kernel	Boston	Breast	Census	Contra	Image	Iris
Sigmoid	0.789	0.963	0.592	0.452	0.904	0.904
Gaussian	0.788	0.972	0.657	0.500	0.906	1.00
–	KDD99	Liver	Pima	Shuttle	Thyroid	Wine
Sigmoid	0.554	0.616	0.707	0.637	0.784	0.948
Gaussian	0.589	0.704	0.713	0.607	0.807	0.952

Values less (more) than unity imply a model with a lower (higher) count of CMGE elements than SVM elements. In eight of the twelve cases, CMGE models had a lower count, whereas in four cases the SVM solution utilized less. However, the factor by which CMGE solutions where deemed simpler was up to an order of magnitude in three cases, and over three orders of magnitude in one case (Census).

The underlying theme in Figure 4-4 is that the SVM 'simplicity' is generally correlated with the smaller data sets; Breast, Image, Iris, Liver, and Wine corresponding to the 5th, 3rd, 1st, 4th, and 2nd smallest data sets respectively. However, after this it more difficult to identify specific second order relations; with the easier Breast, Iris and Wine data sets having a strong preference for an SVM model from a simplicity perspective, but Breast and Wine benefiting from the more complex CMGE model in terms of classification. At the other extreme, the CMGE models that were generally simpler than the SVM solutions returned both weaker and stronger classification scores than the corresponding SVM models (Census and KDD99 classification performance was generally weaker, whereas Shuttle and Thyroid were generally stronger).

Computational Requirements

Computational requirements of each model are summarized for the Thyroid, Shuttle, KDD99 and Census data sets on account of the relative increments associated with the training partition; that is approx. {3,700, 44,000, 140,000, 200,000} exemplars respectively. Moreover, we also detail this characteristic in terms of the duration necessary to conduct a 'typical' run and that to build all the models necessary to make the performance evaluation. This implies fifty models in the case of CMGE and six models in the case of the SVM (two kernels by three regularization parameters). Note, this is summarized as a multiple of the typical run time, some of the SVM regularization parameters resulting in instability, thus runs of over ten hours. From Figure 4-5 the trade off between the SVM and CMGE approaches is again immediately apparent. The constant size

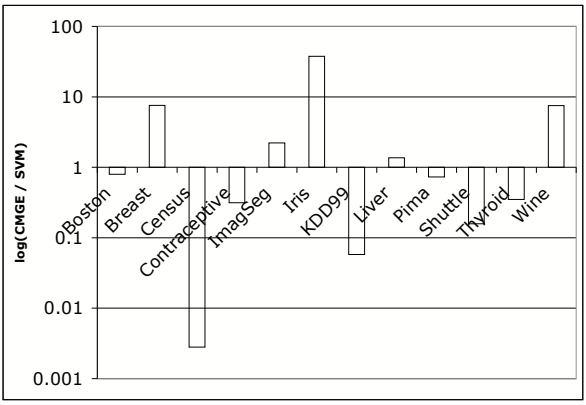


Figure 4-4. SVM performance point normalized CMGE. 'log' ratio of normalized CMGE instruction count versus SVM support vector count under Gaussian kernel. Values less (greater) than unity imply a simpler CMGE (SVM) solution.

of the point population and archives effectly decouple the CMGE model from the specific size of the training data set. When combined with the ability to class-wise introduce early stopping (care of the Pareto front behavior), there is little increase in computation time when training on tens of thousands of exemplars to hundreds of thousands. Conversely, the LIBSVM implementation of the SVM model results in an increasing computational overhead as the training exemplar count increases, where this is also reflected in the strong correlation between SVM model, complexity and size of the training data partition.

5. Conclusions

Advances in areas such as Evolutionary Multi-objective Optimization (EMO) and competitive coevolution are providing new opportunities for addressing credit assignment in GP. Specifically, the proposed CMGE model scales to large data sets using a competitive coevolutionary model in combination with a class-wise balanced uniform sampling heuristic (Weiss and Provost, 2003). A local Gaussian wrapper operator is employed to delimit exactly what subset of exemplars and individual responds to. This insight enables us to evolve the region over which the Gaussian is built, thus associating it with the most dense set of exemplars on the *gpOut* axis. This provides the first mechanism for establishing problem decomposition. EMO evaluation is then very efficient, as

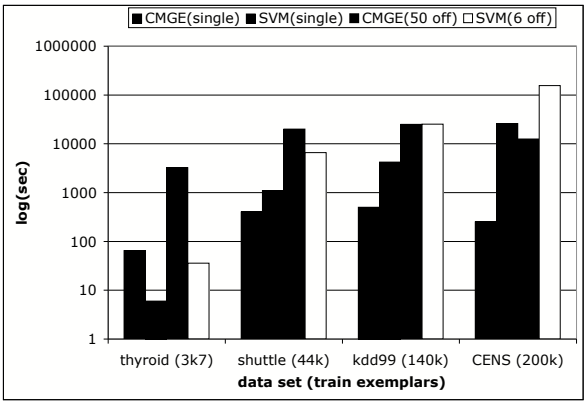


Figure 4-5. SVM and CMGE CPU training time in log seconds. Performance is expressed as a tuple: CMGE and SVM time for a typical run; CMGE and SVM time for 50 and 6 runs respectively.

evaluation is only performed over the exemplars explicitly associated with the region of the Gaussian. One EMO objective explicitly rewards behavior that minimizes the overlap of exemplars mapped to the Gaussian; the second mechanism for establishing problem decomposition. However, a second insight is to 'bootstrap' this objective, by evaluating it against the individuals contained in the archives built during competitive coevolution. Thus, the class-wise archive produced by the competitive coevolutionary model identifies the team of learners; whereas EMO drives the mechanism by which the diversity of learners is maintained, and class-wise early stopping is established.

The strength of the proposed CMGE model naturally lies in the ability of the model to decompose problems such that multiple models support the classification of a single class. The decomposition is entirely an artifact of the evolutionary model. This is most apparent in the general simplicity of the solutions, as compared to the SVM model. Moreover, this is also achieved without sacrificing the classification performance; with data set preferences being returned for both CMGE and SVM solutions. Needless to say, alternative SVM models are available that might result in a different relative weighting of performance factors. A case in point being SVM models that are based on an active learning strategy to explicitly address scaling to large data sets (Bottou et al., 2007). Needless to say, such models introduce a performance trade off in terms of model accuracy; whereas the benchmarking comparison conducted

here was designed to contrast the relative merits of a known strong classification implementation (LIBSVM) versus an evolutionary model designed to address weaknesses in canonical GP. An additional evaluation of the problem decomposition property of CMGE in terms of GP competitive coevolution versus the cooperative-competitive teaming model of CMGE is made in (McIntyre and Heywood, 2008) and (McIntyre, 2007).

Future work will consider the utility of the CMGE framework to problem domains with large feature spaces. The intuition being that as the CMGE model is able to identify intra-class as well as inter-class models, it is now explicitly possible to decouple features appropriate for models at the intra-class level, rather than having to tie feature selection to representation for all class consistent classification (as in one-classifier-per-class frameworks). Such a property could be of particular importance to bio-informatic and document or multi-media classification/ categorization domains in which the feature spaces are both very large, but also sparse and multi-labelled. Results on the data sets considered in this work have been shown to support the above hypothesis for strong intra-class feature subset identification by intra-class classifiers (McIntyre, 2007).

Acknowledgements

The authors gratefully acknowledge the support of PRECARN, NSERC Discovery, MITACS, and CFI New Opportunities programs; and industrial funding through the Telecom Applications Research Alliance (Canada), and SwissCom Innovations AG (Switzerland).

References

Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., editors (2007). *Large-Scale Kernel Machines*. MIT Press.

Chang, Chih-Chung and Lin, Chih-Jen (2007). *LIBSVM: a library for support vector machines*. http://www.csie.ntu.edu.tw/čjlin/libsvm.

Chiu, S. L. (1994). Fuzzy model identification based on cluster estimation. *Journal of Intelligent and Fuzzy Systems*, 2:267–278.

de Jong, E. D. (2004). The incremental pareto-coevolution archive. In *Proceedings of the Genetic and Evolutionary Computation Conference*, number 3102 in Lecture Notes in Computer Science, pages 525–536. Springer.

Drummond, C. (2006). Machine learning as an experimental science (revisited). In *AAAI Workshop on Evaluation Methods for Machine Learning*, pages 1–5. WS-06-06.

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005). Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918.

Harper, R. and Blair, A. (2005). A structure preserving crossover in Grammatical Evolution. In *IEEE Congress on Evolutionary Computation*, volume 3, pages 2537–2544.

Japkowicz, N. (2006). Why question machine learning evaluation methods? In *AAAI Workshop on Evaluation Methods for Machine Learning*, pages 6–11. WS-06-06.

Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–450.

Kumar, R. and Rockett, P. (2002). Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution. *Evolutionary Computation*, 10(3):283–314.

Lim, T.-S., Loh, W.-Y., and Shih, Y.-S. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40:203–228.

McIntyre, A. R. (2007). *Novelty Detection + Coevolution = Automatic problem decomposition: A framework for scalable Genetic Programming classifiers*. PhD thesis, Dalhousie University, Faculty of Computer Science. http://www.cs.dal.ca/m̃heywood/Thesis.

McIntyre, A. R. and Heywood, M. I. (2007). Multi-objective competitive coevolution for efficient GP classifier problem decomposition. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1930–1937.

McIntyre, A. R. and Heywood, M. I. (2008). Cooperative problem decomposition in pareto competitive classifier models of coevolution. In *Proceedings of the European Conference on Genetic Programming*, number 4971 in Lecture Notes in Computer Science, pages 289–300. Springer.

O'Neill, M. and Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358.

Platt, J.C. (1998). Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods – Support Vector Learning*, B. Scholkopf et al., (eds), pages 185–208.

Provost, R., Fawcett, T., and Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In *International Conference on Machine Learning*, pages 43–48.

Weiss, G. M. and Provost, R. (2003). Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354.

Chapter 5

GENETIC PROGRAMMING WITH HISTORICALLY ASSESSED HARDNESS

Jon Klein[1] and Lee Spector[1]
[1]*Cognitive Science, Hampshire College, Amherst, MA USA 01002-3359*

Abstract We present a variation of the genetic programming algorithm, called Historically Assessed Hardness (HAH), in which the fitness rewards for particular test cases are scaled in proportion to their relative difficulty as gauged by historical solution rates. The method is similar in many respects to some realizations of techniques such as implicit fitness sharing, stepwise adaptation of weights and fitness case selection, but the details differ and HAH is generally simpler and more efficient. It also leads to different generalizations. We present results from large-scale, systematic tests of HAH and we also discuss the technique in terms of the alternative explanations that it supports for the efficacy of implicit fitness sharing and related methods.

Keywords: historically assessed hardness, implicit fitness sharing, push, pushGP

1. Introduction

Some problems are harder than others. Within the context of a single problem, some test cases are generally harder than others. Good performance on the hard cases is probably more valuable, all other things being equal, than good performance on the easy cases. How can this differential value be incorporated into the reward function of a machine learning system? Will the incorporation of differential rewards based on problem/case difficulty improve the system's learning performance?

In this chapter we address these questions in the context of genetic programming (Koza, 1992), in which the reward function is called a "fitness function" and test cases are called "fitness cases." We provide a simple mechanism for giving greater fitness rewards for good performance on harder fitness cases,

and we describe the mechanism's impact on performance and relation to other techniques.

"Hardness," however, is not itself a simple concept. Some of the hardness of a fitness case may be intrinsic to the problem that we are trying to solve, or at least to the relation between the problem and the problem-solving tools provided by the genetic programming system. For example, suppose that we are working on a symbolic regression problem, trying to discover a formula that explains a data set, and that the correct answer has a form like:

$$y = \begin{cases} x + 0.4 \text{ if } x > \pi \\ x - 0.4 \text{ if } x < -\pi \\ 1.07x^{11} - 11x^{\pi} + \pi x \text{ if } -\pi \leq x \leq \pi \end{cases}$$

Clearly there is a sense in which the cases with x values between $-\pi$ and π are harder, but is this really an intrinsic property of the problem? Suppose that the genetic programming function set includes standard arithmetic operators, a conditional control structure, and a general exponentiation function, but that the available constants are just those in the set:

$$\{\pi, -\pi, 1.07, 11\}$$

Note that this set does not include the constant 0.4 or "ephemeral random constants" that could produce 0.4. Furthermore, while it is possible to construct 0.4 using the provided constants and standard arithmetic operators it is not trivial to do so. It therefore seems likely that it will actually be *easier* to produce formulae for cases with x values between $-\pi$ and π. So at least some of the hardness of a fitness case may be due to the match between the problem and the functions and constants that are available to the genetic programming system. Whether or not there is a meaningful sense in which fitness cases have "intrinsic" hardness independent of these factors is a complex question that we will not address here, but clearly there are components of hardness that derive from the conjunction of particular problems with particular constraints on the elements out of which solutions can be constructed.

Other aspects of fitness case hardness will derive from the implementation choices made by the user, including the choice of program representation (for example, whether programs are represented as trees, linear strings, grammatical derivations, etc.), the choice of genetic operators (including various types of mutation and crossover), and choices of parameters (including the proportion of each generation produced by each of the chosen genetic operators). These choices collectively define the ways in which a population of programs can be transformed from generation to generation. In combination with the fitness function they define the fitness landscape that genetic programming can search. On some such landscapes programs that perform correctly on particular fitness

cases will be more or less accessible than they are on other such landscapes; therefore, within the context of particular implementation choices some fitness cases will be harder or less hard than they are in the context of other implementation choices.

A final meaning of hardness grows out of a population's current distribution on a fitness landscape, which in turn determines how easily the population can move towards a solution. For example, it is widely accepted that genetic diverisity is a beneficial trait of a genetic programming population because a diverse population is likely to achieve a more thorough exploration of the fitness landscape. Likewise, a population that is mostly centered around steep local maxima is likely to perform worse than a population of equal (or even worse) average fitness that is positioned better relative to the desired solution. This form of hardness is not intrinsic to a problem and changes dramatically across and even during genetic programming runs. Because of its unpredictability and dynamic nature, this form of hardness may be difficult to counteract using non-dynamic strategies such as changing a problem's representation or modifying a run's parameters.

Hardness based on the population distribution derives in large part from "choices" made by the random code generator at the start of a genetic programming run and from other partially-random events such as choices of crossover points and new code produced by mutation. Such events can affect the hardness of particular fitness cases in several ways. For example, if particular functions or constants that are needed for some cases are rare in the initial population of a run then those cases may appear to be harder in the context of that run. Similarly, an initial population that contains a large number of individuals that do well on a particular subset of the fitness cases may lead to subsequent generations dominated by individuals that "specialize" in these cases, making progress on the other cases more difficult.

Our goal in the present project is to develop a method that focuses a genetic programming system's effort on the hard cases, regardless of the nature or source of the underlying hardness. We want to give more rewards for doing well on the hard stuff—for performing better on the more challenging fitness cases—regardless of whether the hardness is due to intrinsic difficulty, representation and implementation decisions, historical "accidents" in a particular run, or combinations of these and other factors. We have accomplished this by developing a technique in which hardness is gauged historically, on the basis of solution rates in the current run. This is easy to do and it captures, in some sense, all of the sources of hardness described above. As we show below, it also seems to produce significant improvements in genetic programming performance, at least in the contexts in which we have tested it.

Some aspects of our new technique, when viewed at the implementation level, are similar to aspects of other techniques such as "implicit fitness sharing"

(sometimes also called "phenotypic fitness sharing"), "stepwise adaptation of weights," and various methods for selecting fitness cases. But these techniques all differ from our new technique both conceptually and with respect to their potential extensions. We discuss the relations between these techniques below, after the presentation of our new method.

2. Historically Assessed Hardness

Our new method, genetic programming with Historically Assessed Hardness (HAH), involves the scaling of values for individual fitness cases based on their empirically determined difficulty. As in much of the literature, we phrase our discussion here in terms of fitness values that are based on errors, and lower values (derived from smaller errors) are better than larger values (derived from higher errors). In this context, most traditional genetic programming systems compute the overall fitness F_i of individual i for n fitness cases as the sum of the absolute value of individual error $|e|$ for each case c; we call this the "unscaled summation" method:

$$F_i = \sum_{c=0}^{n} |e_c|$$

In HAH we differentially scale the individual terms in this summation, and there several ways in which this can be done. In all of the methods that we have studied the scaling function involves a historically assessed *solution rate* for the individual case, designated here as s_c. This solution rate can itself be calculated in several ways; those that we have tested are:

Previous Generation s_c is the fraction of the individuals in the previous generation that produced the correct answer for fitness case c.

Current Generation s_c is the fraction of the individuals in the current generation that produce the correct answer for fitness case c. Note that this requires that the scaling be performed in a second pass through the current generation, after errors have been calculated for all individuals. Because of the extra pass required, we do not advocate the use of this method, but we include it in our experiments for the sake of comparison against implicit fitness sharing.

Cumulative s_c is the fraction of all previously-evaluated individuals in the current run that produced the correct answer for fitness case c. We base this on the errors of all individuals in all previous generations of the current run.

Cross-Run s_c is the fraction of all individuals in a previous, independent run of genetic programming that produced the correct answer for fitness case c.

The independent run (from which the rates are derived) is conducted using the unscaled summation (standard) method for fitness case combination.

However the solution rates (s_c values) are calculated they can be used in a variety of ways to scale the individual errors for each case. We have explored two such methods:

Difference In this method the solution rates, which run from 0 to 1, are subtracted from 1.01 and the product of this difference and the raw error value is used as the scaled value:

$$F_i = \sum_{c=0}^{n} (1.01 - s_c)|e_c|$$

The difference is calculated from 1.01 rather than 1.0 to avoid producing a scaled error of 0, which would be interpreted as success in the rare (but possible) circumstance that e_c is non-zero and s_c is 1.

Quotient In this method the solution rate is multiplied by the population size and divided into the the raw error:

$$F_i = \sum_{c=0}^{n} \frac{|e_c|}{1 + (s_c * P)}$$

The "+1" in the denominator prevents division by zero.

In both of these methods the error for each fitness case is scaled in a way that makes it larger when the solution rate for the case is smaller and smaller when the solution rate for the case is larger. In other words, errors count more (and therefore good performance is rewarded more) for cases that are harder.

3. Related Techniques

Historicallly Assessed Hardness is similar in many respects to a number of existing and well-studied techniques in evolutionary computation, including implicit fitness sharing, stepwise adaptation of weights, and other methods for the selection or weighting of fitness cases such as boosting and bagging (Iba, 1999). Given certain parameters, constraints and problem types, HAH may even be identical to some realizations of these methods.

Broadly stated, what these methods have in common is the scaling of fitness or error values for individual fitness cases based on some heuristic. The details of the methods differ: the scaling values may be computed dynamically at each generation, or at discrete points within a run; the scaling values may be continuous, or they may be binary (e.g. some techniques for fitness case selection can

be described as using a scaling of 0.0 for non-active fitness cases); the scaling values may be computed based on a population's current performance, or on other heuristics.

A full comparison of HAH to all related methods is beyond the scope of this chapter, but we do compare it to a few of its closest relatives in the following sub-sections. For these, and for the others of which we are aware, the bottom line appears to be that while there are overlaps among the techniques each leads to different generalizations and HAH is the simplest method to implement.

Implicit Fitness Sharing

Fitness sharing is a technique for improving the performance of evolutionary computation systems, ostensibly by promoting diversity among the evolving population. In the original formulations of fitness sharing (Deb and Goldberg, 1989) fitness payoffs were reduced in densely populated regions of the search space by performing genotypic analysis and by sharing a local fitness budget among genotypically similar individuals (which collectively form a "niche"). This approach explicitly addresses population diversity at the expense of potentially costly genotype analysis to determine the niches (potentially $O(N^2)$) (Sareni and Krähenbühl, 1998). Further refinements of fitness sharing address concerns raised by the difficulty of genotypic analysis by assigning niches based on *phenotypic* analysis of the similarity of fitness profiles over all test cases. Phenotypic analysis may lead to simpler pairwise analysis of individuals, but it still leads to combinatorial problems in determining niches.

Implicit fitness sharing (Smith et al., 1993; McKay, 2001) foregoes the explicit computation of genotypic or phenotypic niches by divvying up fitness rewards for individual fitness cases based on the number of individuals that produce the same prediction. Although this avoids the need for the explicit computation of differences between individuals, it does require population-wide statistics to be calculated in a multi-pass fashion: the number of individuals sharing the fitness reward is not known until all individuals have been evaluated, so a second pass must be made through the population to compute fitness values. In the terminology of Section 2, implicit fitness sharing, at least when applied to Boolean problems, is nearly equivalent to HAH with error rates determined from the current generation and individual errors combined using the quotient method. Note, however, that this is a relatively complicated version of HAH to implement because it requires multiple passes through the population in order to make use of the statistics of the current generation. The other versions of HAH all differ from fitness sharing at the implementation level and are all simpler and more efficient to implement than implicit fitness sharing. In fact, when the "cross-run" version of the technique is used it is not necessary to compute any run-time population statistics at all, and although this

version is far less effective than the others it nonetheless sometimes provides a benefit over traditional genetic programming.

The differences in the implementations of HAH and implicit fitness sharing stem from the fundamental differences in the theoretical underpinnings of the techniques. Fitness sharing was motivated by a desire to maintain population diversity, while HAH is intended to focus search on the difficult parts of a problem. Sometimes these goals overlap, and in some cases both goals may be served by the same modification to the standard genetic programming technique. But even in such cases the differences in theoretical underpinnings lead to different ideas for extension and generalization and to different explanations of the successes or failures of the techniques in particular circumstances.

A somewhat subtle difference between HAH and implicit fitness sharing, which nonetheless has practical implications, concerns the way in which fitness is shared between individuals. HAH scales fitness rewards/penalties according to the proportion of individuals that produce the correct answer, while fitness sharing divides fitness values among all individuals with the *same* answer, correct or incorrect. For problems with Boolean outputs, in which all incorrect answers for a particular fitness case are necessarily identical, this distinction has no practical impact. For problems in which different incorrect answers may receive different fitness penalties, however, fitness sharing and HAH differ significantly. HAH, as defined above, will still scale all fitness penalties (including penalties for suboptimal solutions) by a case's overall solution rate, while fitness sharing will divide fitness rewards based on the number of individuals producing the same result. For individuals that belong to groups that do not produce the optimal answer, their scaling factor with fitness sharing will not be equal to the fitness case solution rate.

Another way to think about the difference between fitness sharing and HAH is that fitness sharing schemes typically treat fitness rewards as zero-sum resources that are divided up equally among all individuals that produce the same response. HAH differs philosophically from fitness sharing on this point: in HAH fitness rewards or penalties are scaled in order to promote the solution of difficult problems, not to treat fitness as a zero-sum resource. Treating fitness as a zero-sum resource leads to an extreme focus on promoting population diversity, even at the expense of high-quality individuals. This can lead to pathological cases in which an individual that finds only a few correct responses can be awarded a better fitness than individuals that perform far better but also produce correct outputs on fitness cases that are solved by many other individuals. Because implicit fitness sharing does not use any actual (i.e., explicit) measure of similarity between individuals, there is no assurance that the individuals sharing fitness are actually similar, either genotypically or phenotypically (because the fitness reward sharing is done on a per-fitness-case basis, not accounting for an individual's entire phenotypic profile).

Stepwise Adaptation of Weights

Stepwise Adaptation of Weights (SAW) is a related technique which modifies a fitness landscape by scaling an individual's error values based on a set of evolving weights. The technique was originally applied to constraint satisfaction problems in which the weights were applied to individual constraints (Eiben and van Hemert, 1999), but has also been applied to genetic programming (and symbolic regression in particular) by applying the weights to individual fitness cases (Eggermont and van Hemert, 2000).

When applied to symbolic regression problems, SAW resembles HAH in that weights are modified according to performance on a set of fitness cases, but the details of how the weights are computed differ greatly between the techniques. In a typical SAW implementation, weight adjustments are performed according to the performance of the best individual in the population, and only once every several generations (5-20 in the work on symbolic regression, but up to 250 in the original work with constraint satisfaction problems). Weights are initialized to $w_i = 1$ and are updated by adding a constant value if the error for fitness case i is non-zero. An alternative weight updating scheme, dubbed precision SAW, updates the weights in proportion to the fitness case error value itself (Eggermont and van Hemert, 2000) as is done with HAH, although the weight changes in SAW are cumulative while in HAH the weights are recomputed at each evaluation.

Because there is a sense in which, given certain parameters and assumptions, HAH is similar to both implicit fitness sharing and SAW, it is reasonable to suggest that SAW can be considered a form of fitness sharing and vice versa. It seems at first glance counterintuitive that SAW, which uses only the best individual in a population, could be considered a special case of fitness sharing, which depends on measures of phenotypic or genotypic similarity between individuals. However, if one considers the notion that the genes of the best individuals in a population are more likely to be over-represented relative to less successful individuals, then using only the performance of the best individual in a population may be a reasonable (though greatly simplified) proxy measure of the dominant phenotypes in the population.

Fitness Case Selection

Another broad area of research related to HAH concerns fitness case selection methods. These use various heuristics for choosing subsets of the fitness cases for fitness testing. Many of these are intended to reduce the number of fitness cases so that fitness evaluations can be run more quickly, and many use subsets based on random selection. Dynamic Subset Selection (DSS) expands on the notion of fitness case selection by incorporating fitness case difficulty as a criterion for fitness case selection (Gathercole and Ross, 1994). More recently,

Topology Based Selection (TBS), a heuristic based on similarity between fitness cases (based on co-incidence of their solution) has proven to be even more effective than DSS at improving the performance of GP on regression and classification problems (Lasarczyk et al., 2004).

HAH does not deal explicitly with the selection of fitness case subsets for testing, but fitness case selection-like effects do sometimes emerge with HAH, particularly when the population evolves to the point at which certain fitness cases are fully solved. However, the most notable benefit of fitness case selection methods, namely a reduction in the number of fitness cases, is not present in HAH. So while HAH, like some variations of fitness case selection, may improve the performance of GP by changing the fitness landscape, HAH is not useful in reducing computation time required for individual fitness evaluations.

4. Experiments

We examined the performance of Historically Assessed Hardness in the evolution of a solution to the n-parity problem, in which a program must determine whether a variable-length list of inputs has an even or odd number of "true" values. We chose this problem, which we have studied previously, as an example of a moderately difficult classification problem.

We used the PushGP genetic programming environment (Spector, 2001; Spector and Robinson, 2002; Spector et al., 2004; Spector et al., 2005) integrated with the breve simulation environment (Klein, 2002). Push is a multi-type stack-based programming language developed for evolutionary computation and which allows the evolution of novel control structures through explicit code and control manipulation. These control structures are of particular interest for problems such as n-parity which require iterative or recursive solutions, especially in the absence of explicit iterator functions, as in the current work.

The instruction set used is found in Table 5-1. The instruction set includes standard stack manipulation instructions for all types, integer math instructions, Boolean logic instructions and instructions for explicit manipulation of code and program control (via the EXEC stack, which contains a list of all instructions to be executed).

Notably absent from the instruction set we used are explicit iterator instructions that Push supports. These iterator functions (such as CODE.DOTIMES and EXEC.DO*COUNT, among others) allow for the construction of explicit loops, and the absence of these instructions makes finding solutions considerably more difficult. In place of iterator functions are special combinator functions (Y, K and S) which manipulate the EXEC stack and which can be helpful in the evolution of iterative or recursive behaviors that use novel, evolved control structures (Spector et al., 2005).

Table 5-1. Instruction set for tests of Historically Assessed Hardness on the n-parity problem. Full documentation for these instructions is available in the Push language specification (Spector et al., 2004).

Types	Instructions
Integer instructions	FROMBOOLEAN $<$ $>$ MAX MIN % / * - + STACKDEPTH SHOVE YANKDUP YANK = FLUSH ROT SWAP POP DUP RAND
Boolean instructions	FROMINTEGER NOT OR AND STACKDEPTH SHOVE YANKDUP YANK = FLUSH ROT SWAP POP DUP
Code instructions	FROMNAME FROMBOOLEAN FROMINTEGER DO SIZE LENGTH EXTRACT INSERT NTHCDR NTH APPEND LIST NOOP IF DO* CONS CDR CAR NULL ATOM QUOTE STACKDEPTH SHOVE YANKDUP YANK = FLUSH ROT SWAP POP DUP
Name instructions	QUOTE STACKDEPTH SHOVE YANKDUP YANK = FLUSH ROT SWAP POP DUP
Exec instructions	Y S K IF STACKDEPTH FLUSH POP DUP

Table 5-2. Parameters used with PushGP for the n-parity problem runs.

Population size	5000
Evaluation limit	2000
Mutation	40% fair mutation 5% deletion mutation
Crossover	40%
Copy operator	15%
Tournament selection size	7
Generation limit	300
Fitness cases	64 randomly generated lists of between 8 and 12 Boolean values (the same randomly generated fitness cases were used for all runs)

Table 5-3. Results of various fitness scaling methods on the n-parity problem, using the parameters in Table 5-2 and the instructions in Table 5-1, sorted by success rate ("% Succ."). Abbreviations used in the method descriptions: Prev = Previous Generation, Cum = Cumulative, Cross = Cross-Run, Curr = Current Generation, Diff = Difference, Quot = Quotient. Note that "Curr/Quot" is equivalent to implicit fitness sharing for the n-parity problem (because it is a Boolean problem; see text). Computational effort, described by Koza, represents the cumulative effort required to solve a problem, taking into account both solution rate and generations required to find a solution (Koza, 1992).

Method	Succ. /Runs	**% Succ.**	Av. Succ. Gen.	Comp. Effort	Mean Best Fitness
Prev/Diff	180/233	**77.253**	125.622	1,800,000	0.9674
Curr/Quot	169/234	**72.222**	111.520	1,752,000	0.9957
Prev/Quot	162/234	**69.230**	126.456	2,124,000	1.0206
Cum/Diff	157/232	**67.672**	129.713	2,208,000	1.0734
Cum/Quot	140/234	**59.829**	137.671	2,880,000	1.0721
Cross/Quot	57/234	**24.358**	157.140	9,288,000	1.5335
Cross/Diff	49/234	**20.940**	126.448	4,304,000	1.5678
Unscaled	48/234	**20.512**	120.395	8,282,000	1.6515

A complete list of the parameters used during the experiment is shown in Table 5-2. We conducted 234 runs of each technique, with 4 runs lost to miscellaneous software and system problems.

5. Results

The results of our runs are shown in Table 5-3. The HAH configuration in which solution rates were obtained from the previous generation and errors were scaled using the difference method produced the most solutions overall and the lowest (best) mean best fitness. Several other configurations also performed quite well; in fact all of the HAH configurations aside from the cross-run configurations significantly out-performed the standard, unscaled method of combining errors on individual fitness cases. The savings provided by each of these methods, in terms of computational effort, was about 75%.

The configuration that is essentially equivalent to implicit fitness sharing ("Curr/Quot") was among the best; indeed it had the lowest average success generation and the lowest computational effort. But the implementation of this method (using the statistics of the current generation) is somewhat more complex than that of the others, so it is not necessarily the best choice. It is also important to bear in mind that none of these methods is strictly equivalent to implicit fitness sharing when applied to non-Boolean problems.

The performance of the cross-run HAH methods, while much worse than that of the other HAH methods, is interesting in several respects. In terms

of computational effort one of these methods ("Cross/Diff") performed much better than the standard ("Unscaled") method, but the other of these methods ("Cross/Quot") did somewhat worse than standard. However, both of these methods out-performed the standard method both in terms of overall success rate and in terms of mean best fitness. The high computational effort of "Cross/Quot" is attributable to the relatively large number of generations it required to achieve success. Hence there are several senses in which even these methods, which used solution rates from independent runs as the basis for their scaling of errors, out-perform the standard method. To the extent that this is true one can infer that the improvements are due to aspects of case "hardness" that transcend the circumstances of a particular genetic programming run.

6. Conclusions and Future Work

We have presented a family of easily implemented methods that are designed to focus the effort of a genetic programming run on harder fitness cases, where hardness is determined empirically from solution rates during a genetic programming run. We have shown that these methods can produce significant improvements in problem-solving performance for a small investment of programmer effort. One of our methods turns out to be nearly identical, at the implementation level, with one type of fitness sharing that has been described in the literature (implicit fitness sharing), but the rationale for our methods is completely different from that offered in discussions of fitness sharing. Whereas fitness sharing is focused on the maintenance of diversity in a population, our Historically Assessed Hardness (HAH) methods are based on the idea of rewarding good performance on difficult problems. This difference in rationale leads to variations and extensions of our technique that are different than those that arise naturally from fitness sharing approaches.

We believe that HAH can have significant utility in a wide range of application areas. In fact, the exploration documented in this chapter was launched because our use of a form of HAH helped us to achieve new results in an application-oriented project (Spector et al., 2008).[1] In that project, however, we did not conduct sufficiently many or sufficiently systematic runs to say anything definitive about the efficacy of the method. Here we have conducted $1,872$ runs on a well-studied problem and shown that the method does clearly have utility.

One direction for further research concerns the application of HAH to non-Boolean problems. As described above, in the context of such problems HAH is less similar to fitness sharing and there are reasons to believe that it will

[1] The formulation used in the prior work was essentially the method that we have described here as "previous generation, difference scaling," although the scaled values ran from 1.0 to 2.0 rather than 0.01 to 1.01.

also be useful here. But we cannot say anything definitive about this without large-scale, systematic testing.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 0308540. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation. Special thanks to Wolfgang Banzhaf, Michael Korns, Loryfel Nunez, Katya Vladislavleva, Guido Smits Malcolm Heywood and Ian Lindsay.

References

Deb, Kalyanmoy and Goldberg, David E. (1989). An investigation of niche and species formation in genetic function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 42–50, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Eggermont, J. and van Hemert, J. I. (2000). Stepwise adaptation of weights for symbolic regression with genetic programming. In *Proceedings of the Twelveth Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'00)*.

Eiben, Gusz and van Hemert, Jano (1999). SAW-ing EAs: Adapting the fitness function for solving constrained problems. In Corne, David, Dorigo, Marco, and Glover, Fred, editors, *New Ideas in Optimization*, pages 389–402. McGraw-Hill, London.

Gathercole, Chris and Ross, Peter (1994). Dynamic training subset selection for supervised learning in genetic programming. In Davidor, Yuval, Schwefel, Hans-Paul, and Männer, Reinhard, editors, *Parallel Problem Solving from Nature III*, volume 866, pages 312–321, Jerusalem. Springer-Verlag.

Iba, Hitoshi (1999). Bagging, boosting, and bloating in genetic programming. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1053–1060, Orlando, Florida, USA. Morgan Kaufmann.

Klein, J. (2002). breve: a 3d environment for the simulation of decentralized systems and artificial life. In Standish, R., Bedau, M. A., and Abbass, H. A., editors, *Proc. Eighth Intl. Conf. on Artificial Life*, pages 329–334. Cambridge, MA: MIT Press.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Lasarczyk, Christian W. G., Dittrich, Peter W. G., and Banzhaf, Wolfgang W. G. (2004). Dynamic subset selection based on a fitness case topology. *Evol. Comput.*, 12(2):223–242.

McKay, R. I. (Bob) (2001). An investigation of fitness sharing in genetic programming. *The Australian Journal of Intelligent Information Processing Systems*, 7(1/2):43–51.

Sareni, Bruno and Krähenbühl, Laurent (1998). Fitness sharing and niching methods revisited. *IEEE Trans. Evolutionary Computation*, 2(3):97–106.

Smith, R., Forrest, S., and Perelson, A. (1993). Searching for diverse, cooperative populations with genetic algorithms.

Spector, Lee (2001). Autoconstructive evolution: Push, PushGP, and Pushpop. In Spector, Lee, Goodman, Erik D., Wu, Annie, Langdon, W. B., Voigt, Hans-Michael, Gen, Mitsuo, Sen, Sandip, Dorigo, Marco, Pezeshk, Shahram, Garzon, Max H., and Burke, Edmund, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 137–146, San Francisco, California, USA. Morgan Kaufmann.

Spector, Lee, Clark, David M., Lindsay, Ian, Barr, Bradford, and Klein, Jon (2008). Genetic programming for finite algebras. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, London. ACM Press. To appear.

Spector, Lee, Klein, Jon, and Keijzer, Maarten (2005). The push3 execution stack and the evolution of control. In Beyer, Hans-Georg, O'Reilly, Una-May, Arnold, Dirk V., Banzhaf, Wolfgang, Blum, Christian, Bonabeau, Eric W., Cantu-Paz, Erick, Dasgupta, Dipankar, Deb, Kalyanmoy, Foster, James A., de Jong, Edwin D., Lipson, Hod, Llora, Xavier, Mancoridis, Spiros, Pelikan, Martin, Raidl, Guenther R., Soule, Terence, Tyrrell, Andy M., Watson, Jean-Paul, and Zitzler, Eckart, editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1689–1696, Washington DC, USA. ACM Press.

Spector, Lee, Perry, Chris, Klein, Jon, and Keijzer, Maarten (2004). Push 3.0 programming language description. Technical Report HC-CSTR-2004-02, School of Cognitive Science, Hampshire College, USA.

Spector, Lee and Robinson, Alan (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

Chapter 6

CROSSOVER AND SAMPLING BIASES ON NEARLY UNIFORM LANDSCAPES

Terence Soule[1]
[1]*Department of Computer Science, University of Idaho, Moscow, ID 83844-1010.*

Abstract In this paper we measure the frequency of program sampling as a function of program size under crossover. Following previous work it is shown that crossover bias produces a Lagrange distribution of programs as a function of size and that in a completely flat landscape this heavily oversamples the smallest programs. However, we find that even an extremely small fitness bias towards larger programs produces some significant changes in the sampling distribution. In the presence of both a fitness bias and a size limit, sampling is relatively uniform across all program sizes. Several stochastic fitness functions are tested, producing qualitatively similar results. These show that a sufficient condition for growth due to the crossover bias is that some larger programs are more fit than any of the smallest programs. Results with size fair crossover show that it also tends towards a Lagrange distribution, albeit more slowly than more traditional forms of crossover. Finally, we discuss what an *ideal* sampling distribution might be.

Keywords: bloat, crossover bias

1. Introduction

Genetic programming is a search process that works by conducting a guided sampling of the search space. In general, we assume that this sampling is guided by information obtained through the fitness function. I.e. the selection process is designed to guide the search towards the most promising areas. However, there are other biases that effect what part of the search space is sampled as well.

Some of these biases are the result of the interaction between selection and other operators. E.g. removal bias, which results in growth – a tendency to sample progressively larger programs (Soule and Foster, 1998). Daida et al. have demonstrated that other biases in the sampling processes leads GP to only

sample programs within a very narrow range of program shapes (Daida et al., 2006; Daida et al., 2004).

Most recently, Dignum and Poli have uncovered another bias inherent in the tree-based GP crossover operation (Dignum and Poli, 2008; Poli et al., 2007; Dignum and Poli, 2007; Poli et al., 2008). (Early results showed a related bias in linear GP (Poli and McPhee, 2001)). They showed both theoretically and experimentally that on a flat fitness landscape, i.e. when there is no net selection pressure, crossover leads to a Lagrange distribution of programs in which most programs are extremely small with only a relatively few larger programs (see Figure 6-2 as an example).

Poli et al. have further hypothesized that this crossover bias may be a cause of code bloat. When selection is applied these very small programs are unlikely to be fit, whereas the larger programs are more likely to be fit. The resulting selection of large programs over small programs produces the crossover bias theory of bloat. One open question is what fitness distributions is sufficient to produce growth under these circumstances. E.g. do the larger programs have to be more fit on average, or is it sufficient for a few larger programs to be more fit than any small program, even if the average large program is no more fit (or is less fit) than the average small program.

In this paper we examine the effect of a minimally non-flat landscapes and the use of size fair crossover on the distribution caused by crossover bias and resulting growth. Two general fitness functions are tested. In the first, the very smallest programs are less fit than larger programs. In the second, a stochastic component is added to the fitness function, making the average fitness larger for smaller programs. These fitness landscapes change the sampling distribution observed by Dignum and Poli in some significant ways, while other features of the distribution remain unchanged. Including a stochastic component in the fitness function so that the smallest programs are not always less fit slows the rate of convergence to the Lagrange distribution, but does not appear to affect the final distribution. Similarly, with size fair crossover the population still appears to tend towards a Lagrange distribution, but much more slowly. Finally, the ideal distribution for effective search is discussed.

2. Experimental Parameters

The goal of these experiments is to examine whether, and if so how, crossover biases GP's sampling of the search space independent of the pressure created by the fitness function. The GP used is in some sense 'generic' there is a single arity 2 non-terminal and a single terminal. (The work by Dignum and Poli shows that using a more realistic non-terminal set with a mixture of arities does not change the fundamental results.) Because the only non-terminal has arity 2 and there is a single root node, the only allowed tree sizes are odd, e.g. trees can be

size 1, 3, 5, etc. A steady state model is used, with standard sub-tree crossover and no mutation (the non-terminals and terminals are effectively placeholders so mutation would be meaningless). Generally, the results are the average of 50 trials (exceptions are noted in the text).

A somewhat significant feature of this work is that the 90/10 rule is used. Previous theoretical analysis concluded that inclusion of the 90/10 rule should not change the basic distribution introduced by crossover bias (Dignum and Poli, 2007). Thus, one goal of this work is to empirically confirm that the 90/10 rule does not affect the general crossover bias.

Several fitness landscapes of increasing complexity are used. The first landscape is completely flat, all individuals have the same fitness. The second landscape has two fitnesses 'good' and 'poor'. All individuals get the good fitnesses except for a small 'hole' for programs of size 1, which get the poor fitness. This reflects the likelihood that for most problems very small programs, e.g. with only 1 node, will be insufficient to solve the problem. More generally, it seems reasonable to assume that very small programs are poor, but that larger programs may be better or worse. Consider the case of symbolic regression. Very small expression trees are likely to be too small to capture the solution, hence they get a poor fitness. However, larger programs may be better or worse than the smallest programs. Thus, for the third fitness function programs with 1 node get the fitness poor, however, larger programs are randomly assigned the fitnesses 'good' (which is better than poor) or 'worst' (which is worse than poor). Two versions of this third fitness function were tested. In the first (listed as fitness 1 in the graphs) larger programs have an equal probability of getting a 'good' or a 'worse' fitness. In the second (fitness 2 in the graphs) larger programs have a 10% chance of getting a 'good' fitness and a 90% chance of getting a 'worse' fitness. Thus, for this fitness function the 'average' fitness of small programs is better than for large programs.

For comparison purposes three different methods of generating initial populations are compared: full with a depth of 4, grow with a maximum depth of 7, and ramped half-and-half with depths of 3, 4, 5, and 6. Results with and without a size limit of 500 are also compared. Table 6-1 summarizes the other parameters that are used in these experiments.

3. Results

Experiment 1 – Flat landscape

Figure 6-2 shows the distribution of program trees for the flat fitness landscape in generation 49, when the initial population is generated using grow with a maximum depth of 7, full with a depth of 4, and ramped half-and-half with depths 3, 4, 5 and 6. As a reference point Figure 6-1 shows the distributions at generation 0. The results for generation 49 shows the Lagrange distribution in

Table 6-1. Summary of the Genetic Program parameters.

Function Set	generic arity two operator
Terminal Set	generic leaf node
Population Size	1000
Crossover Probability	1.0
Mutations Probability	Not used
Selection	3 member tournament
Iterations	50 * 1000 replacement = 49,000
Maximum Tree Size	None or 200 nodes
Elitism	None
Initial Population	Full, grow, and ramped half-and-half (see experiments)
Number of trials	50

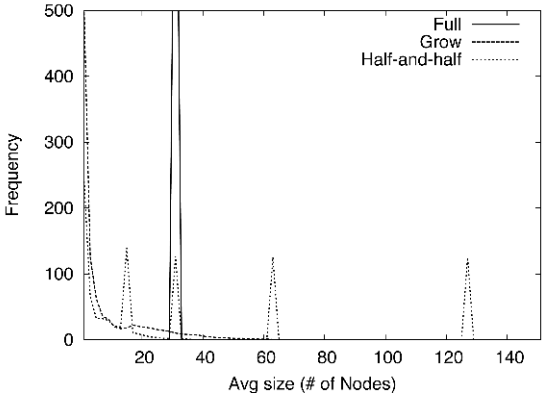


Figure 6-1. Distribution of program trees by size in generation 0 for three common initialization procedures: full, grow, and ramped half-and-half.

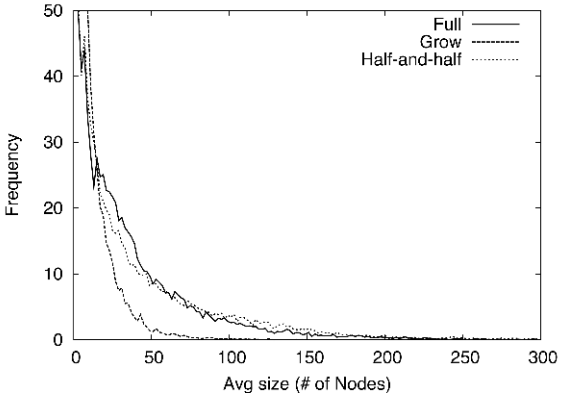


Figure 6-2. Distribution of program trees by size in generation 49 for three common initialization procedures: full, grow, and ramped half-and-half. Variations in the curves are due to difference average programs sizes for the different initialization techniques.

agreement with previous research. This both confirms their results and demonstrates that the use of the 90/10 rule, as is common in GP, does not invalidate their analysis or conclusions. Note that the differences between the distributions in Figure 6-2 are primarily a result of the differenet average program sizes for the different initialization routines.

What is significant about this distribution is that it is heavily biased toward the smallest program trees. Under almost any GP paradigm there will be relatively few unique trees of such small size. Thus, as pointed out by Dignum and Poli, the trees that are in this sub-region are heavily oversampled – wasting significant resources. (Note that because, in this experiment, selection is effectively random, the average size of the programs does not change between generation 0 and generation 49, only the distribution of the programs sizes is changing.)

This sampling distribution is particularly disturbing because it does not match the distribution of programs. In general the number of unique programs grows exponentially with the size of the programs. Thus, ignoring other factors the space of larger programs should be sampled more heavily as that is the larger portion of the search space. Further, for many interesting problems the smallest programs are unlikely to yield an interesting solution, thus heavily sampling the smallest programs is doubly inefficient.

(The one case where the sub-region defined by small trees is arguably not small is in the case of a GP with real valued constants. In this case the range of possible constants is such that there is a significant number of even very small trees. However, even in this case the number of unique tree *structures*, which is arguably the important feature, is very small and thus, heavily over sampled.)

Experiment 2 - Landscape with a hole

To reflect the fact that for most difficult problems the very smallest programs are not very fit a fitness 'hole' for the smallest trees was added to the otherwise flat landscape. Specifically, programs whose size is 1 get the 'poor' fitness and all larger programs get the standard 'good' fitness.

Figure 6-3 shows the distribution of program trees for the modified landscape in generation 49 for trees generated by the grow, full, and ramped half-and-half methods. The distribution is much more uniform then when the fitness is completely flat. The spike in the graph at size 301 represents all programs of size 301 or more and extends to extends to 328 (full), 359 (grow), 418 (ramped half-and-half) for the respective initialization routines. Thus, by generation 49 over one third of the total programs are of size 301 or greater. The smallest programs are clearly no longer *as* heavily oversampled as with the completely uniform fitness function. However, given that the number of unique solutions increases exponentially with tree size, there is still a strong argument that too

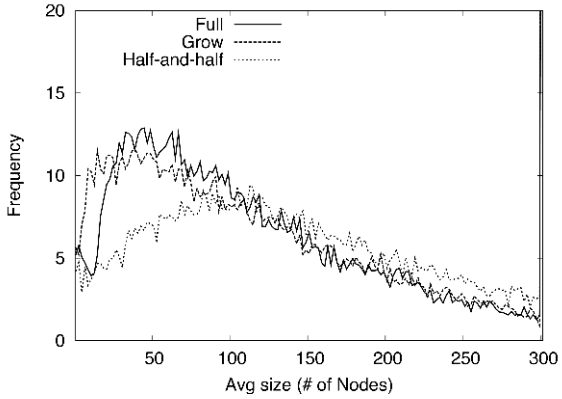


Figure 6-3. Distribution of program trees by size in generation 49 for three common initialization procedures: full, grow, and ramped half-and-half, when trees of size 1 are less fit. All programs of size 301 or greater are represented by the spike at 301, which extends to 328 (full), 359 (grow), 418 (ramped half-and-half) for the respective initialization routines.

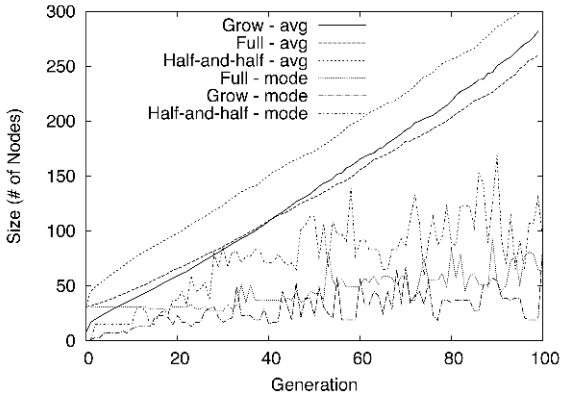


Figure 6-4. Average and mode sizes of program trees for three common initialization procedures: full, grow, and ramped half-and-half, when trees of size 1 less fit. Data from the completely flat landscape (and grow initialization scheme) is included as a reference. Although the landscape is flat for all programs larger than size 3 significant growth is still observed.

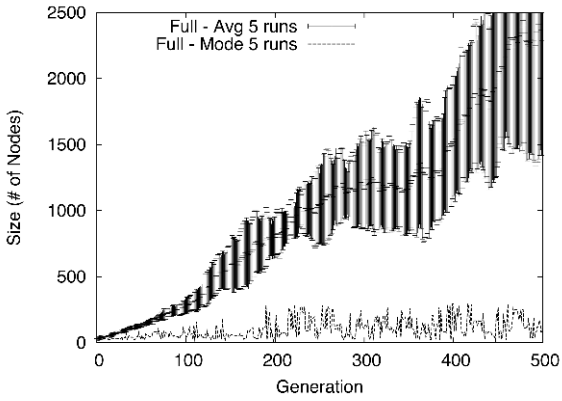


Figure 6-5. Average size of program trees over 500 generations when trees of size 1 are less fit. Only five trials were run.

much of the search is concentrated on the smallest programs, which represent a relatively small proportion of the search space.

The most interesting feature of this graph is that it shows that a very small adjustment to the fitness function - the landscape remains flat for programs larger than 1 node - completely changes the sampling distribution. This results agrees with results reported previously for linear GP (Poli and McPhee, 2001).

Figure 6-4 shows the average and mode sizes for the three initialization routines and Figure 6-5 shows the average size, with standard deviation, and the mode size, for five trials that were extended to 500 generations. Figure 6-5 shows that with the addition of the fitness 'hole' for small programs the overall program size grows fairly rapidly and for many generations. This shows that the influence of lower fitnesses individuals remains quite strong even when the average size of the population is 1500 or more.

Both figures show that although average size grows quite rapidly, the mode size changes relatively slowly. This shows that the change in average size is due to the lengthening tail, rather than a change in the mode program size. This reflects crossover's bias towards the long-tailed Lagrange distribution.

Dynamic Landscapes. Figure 6-6 shows the effect of turning fitness on and off. From generations 0 to 33 fitness is uniform and the distribution moves toward a Lagrange distribution, i.e. the average size remains flat and the mode moves to zero. From generations 33 to 66 the fitness hole is used and the distribution moves from the Lagrange distribution observed with the flat fitness function, to the long tailed distribution observed in Figure 6-3, i.e. the average size grows and so does the mode. After generation 66 the fitness function is switched back to the uniform function and the distribution of program sizes re-

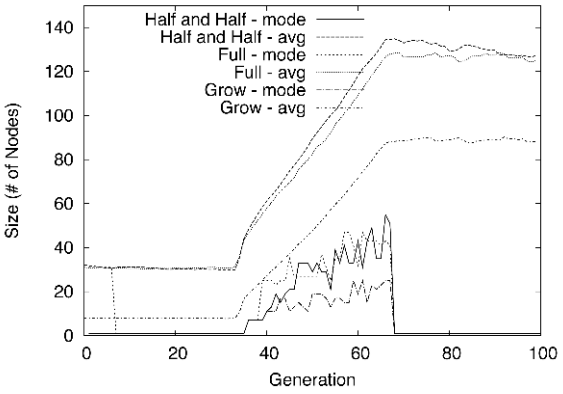


Figure 6-6. Average and mode of program trees with a dynamic fitness. The 'hole' fitness function is turned on only from generations 33 to 66 and the fitness landscape is flat for all other generations. (Trials have been extended to 100 generations.)

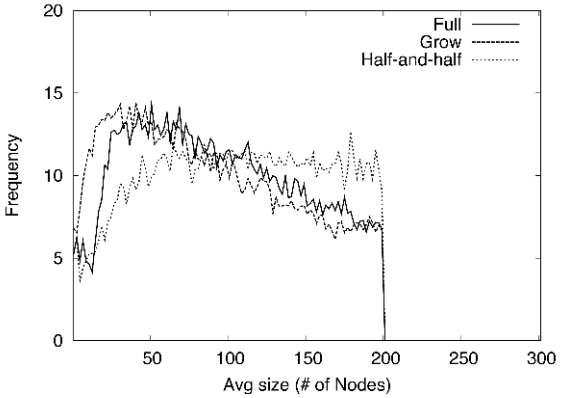


Figure 6-7. Distribution of program trees by size in generation 49 for three common initialization procedures: full, grow, and ramped half-and-half, when trees of size 1 are less fit and a size limit of 200 is imposed. Note that in general sampling is fairly uniform by size.

turns to the Lagrange distribution with a zero mode and a fixed average leading to heavy oversampling of the smallest programs again. This illustrates the idea that crossover, fitness, mutation, and other GP operators have 'desired' distributions (Dignum and Poli, 2007) – when the role of selection is removed the distribution favored by crossover bias is quickly reestablished. This result also introduces the possibility of controlling the population distribution by turning fitness on and off.

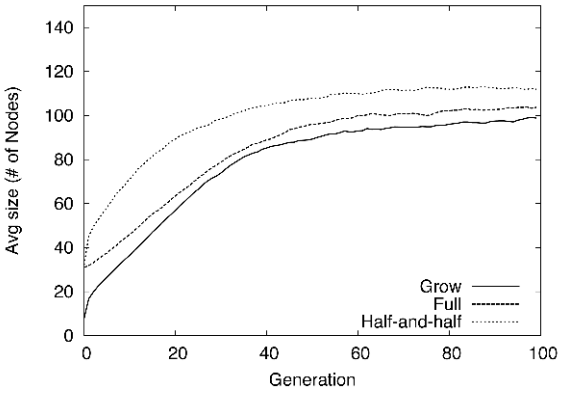


Figure 6-8. Average sizes of program trees for three common initialization procedures: full, grow, and ramped half-and-half, when trees of size 3 or smaller are less fit, and a size limit of 200 is imposed. Overall growth is limited.

Experiment 3 - Landscape with a hole and a size limit

In this experiment a size limit of 200 was added to the genetic program, if an offspring exceeds this size it's discarded and its parent is kept instead. This approach is one of several related techniques that are commonly used to control bloat. Figure 6-7 shows the resulting distribution of programs by size at generation 49. The overall distribution is fairly flat and cuts off at the size limit. There is a dip at the very smallest sizes, which reflects the lower fitness of those programs.

Figure 6-8 shows the average sizes for the three initial conditions when the size limit is included. The size limit clearly limits the overall growth rate, as expected, although it does so without significantly modifying the distribution of solutions within the range of allowed lengths, instead it removes the long tail (Figure 6-3 versus Figure 6-7).

Experiment 4 – Stochastic landscapes

For these experiments the landscape has the same 'hole' for programs with one node. Programs with more than 1 node have a probability P of getting a fitness of 'good' and a probability $1 - P$ of getting a fitness 'worse' which is worst than for the small programs. Thus, larger programs are not always better than the smallest programs. Two values of P are used: 0.5 (fitness 1) and 0.1 (fitness 2). In the former case (P = 0.5) the average fitness of the larger programs is equal to the fitness of the smallest programs. In the later case (P = 0.1) larger programs are, on average, less fit than smaller programs.

The results (Figure 6-9) show that with both fitness 1 and fitness 2 the eventual distribution is very similar to the distribution with just the fitness hole (Figure 6-

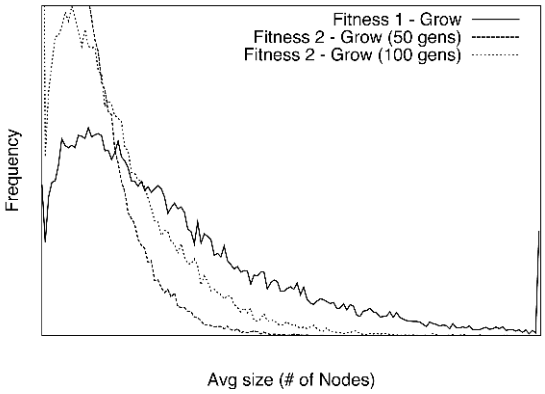


Figure 6-9. Distribution of programs by size at generation 50 (or 100) for the two stochastic fitness function. Initial programs are generated using grow.

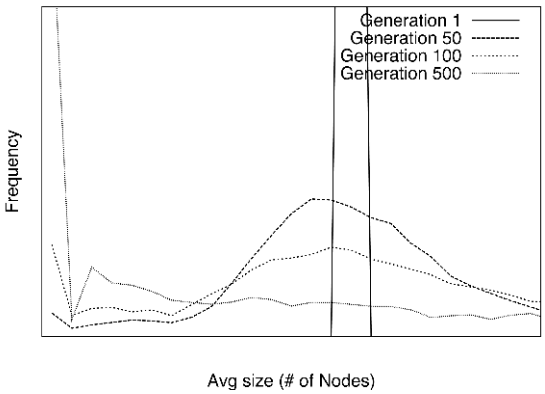


Figure 6-10. Distribution of programs by size at generations 1, 50, 100, and 500 with the flat fitness landscape and size fair crossover. Initial programs are generated using full. In generation 1 all programs are size 31, the size generate using the full initialization routine. The distribution moves toward a Lagrange distribution, although it takes significantly longer than with regular crossover.

3) although it takes many more generations, over 100 with fitness 2, to reach the distribution. Thus, even when the larger programs are not consistently more fit than the smallest programs there is still considerable growth and there is less oversampling of the smallest programs.

Size Fair Crossover

This set of experiments examines the role of the choice of crossover operation in determining the sampling distribution. For these experiments the flat landscape was used with a variation of size fair crossover, which typically

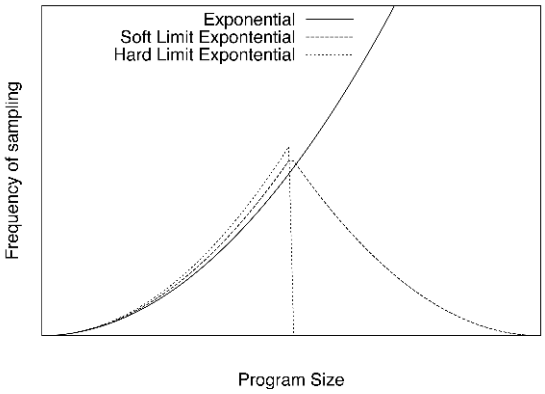


Figure 6-11. Possible target sampling distributions based on the exponential increase on the number of programs of a given size (and the limitations imposed by a finite population size).

diminishes growth (Langdon, 1999). Size fair crossover was implemented as follows:

1. Two parents are selected via tournament selection as in the previous experiments.
2. Two crossover points are selected using the 90/10 rule as in the previous experiments.
3. The sizes of the two branches selected from crossover are calculated.
4. If the size of the larger branch is more than 300% larger than the smaller branch return to step 2; else use the selected branches.

The 300% condition is necessary to allow branches consisting of a single node (which are selected 10% of the time under the 90/10 rule) to be exchanged with branches with 3 nodes.

Figure 6-10 shows the resulting distributions at generations 1, 50, 100, and 500 using the full initialization routine. By generation 500 it is clear the the population is tending to the Lagrange distribution, although the change occurs significantly more slowly than with regular crossover. The deviation for programs of size 3 appears to be due to the interaction of the 90/10 rule, which limits how often single nodes are selected for crossover, and the 300% limit, which limits which branches a branch with 3 nodes can be swapped with.

4. Discussion – Where should we sample?

These results have shown that the crossover operation can and does introduce biases in the sampling of the search space. Perhaps more importantly the results

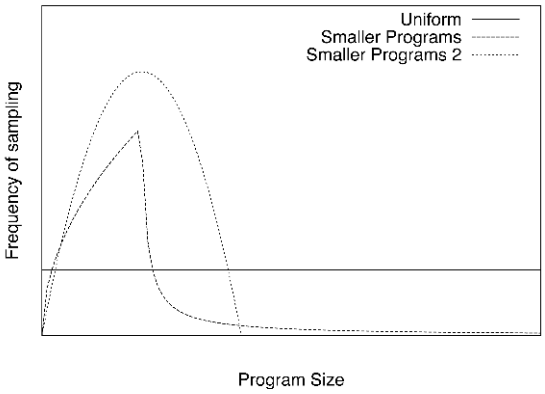


Figure 6-12. Possible target sampling distributions based on the need for compact, general, and/or human-readable programs, or on our current lack of knowledge of ideal sampling distributions.

show that even very small changes either in the GP algorithm, such as the use of a size limit, changes in the fitness function, crossover variations, etc. can significantly affect sampling bias. This suggests that by careful manipulation of the GP algorithm it should be possible to modify the sampling distribution.

This raises the question of what is the ideal sampling distribution. The best answer would be where the solution is, but of course this isn't known *a priori*. Instead we can consider factors that might influence our choice of preferred sampling region:

1. Where solutions are most likely – for most interesting problems the solution is unlikely to consist of very few nodes (e.g. 5 or less). This means that the Lagrange distribution is not the ideal search distribution.

2. The number of potential solutions as a function of size – very generally the number of unique solutions increases exponentially with tree size. Thus, we could attempt to sample proportional to that number. However, in general the very rapid increase in the number of potential solutions as a function of size makes this impractical when sampling relatively large program sizes (Figure 6-11, exponential).

3. The limitations of population size – to account for finite population sizes exponential sampling could be tempered with a size limit. For example, exponential sampling up to some maximum size and no sampling beyond that size. Alternatively a soft limit could be used, exponentially increasing sampling up to some size and exponentially decreasing sampling beyond that limit (Figure 6-11, hard limit exponential and hard limit exponential).

4. The need for general and/or readable solutions – for many practical applications solutions shouldn't overfit and/or must be human readable, both conditions require relatively smaller programs. This suggests using a Gaussian, Gamma, or similar distribution with a mean at an 'ideal' size that balances the likelihood of finding a solution with the need for a small solution (Figure 6-12, smaller programs and smaller programs 2).

5. The need for a general default – given the myriad factors, many still unknown, that potentially affect the ideal distribution we could fall back on something simple, such as a uniform sampling (Figure 6-12, uniform).

Clearly these are just a few of the infinite number of possible sampling distributions. However, there are two points that we should keep in mind. First, there is presumably some ideal sampling distribution that we would like to achieve - although it likely depends on both the problem and our requirements on the solution (for example, the requirement of compact, human-readable solutions). Second, there are many factors that bias, or potentially bias, the sampling distribution, including factors that we are aware of, e.g. removal bias and now crossover bias, factors that we chose to impose, e.g. parsimony pressure or size limits, and possibly still unidentified factors.

5. Conclusions

First, the results independently confirm previous results that standard GP crossover is biased to produce a Lagrange distribution of programs by size. It is also empirically shown that the use of 90/10 crossover doesn't significantly change this distribution.

Second, the results show that although a very small change to the fitness landscape significantly changes the sampling distribution, crossover bias still has a significant influence on the sampling distribution. Specifically if the smallest programs are less fit and the rest of the landscape remains completely flat the resulting distribution of programs is very flat, extends to very large programs, and overall growth is considerable. However, the distribution maintains a very long tail, which accounts for most of the average change in size.

Further experiments showed that as long as some proportion (10% in these experiments) of the larger programs had a higher fitness (and the other 90% had a lower fitness) growth occured. Thus, it appears that a sufficient fitness criteria for the crossover bias theory of growth to apply is that at least some (of the sampled) larger programs have fitnesses that exceed the fitnesses of the (sampled) smaller programs. Note that in these experiments the *average* fitness of the larger programs was less than the average fitness of the smaller programs and growth still occured, albeit slowly.

In general, this and the proceeding research suggests that the focus on bloat may be too limited. Looking at the distribution of program sizes, rather than

just the average size, is better for determining whether the search space is being sampled effectively. Further, the results with size limits suggest that through careful manipulation of bloat control mechanisms, and presumably of the selection mechanisms, it is possible to modify the distribution of programs. Thus, it becomes important to learn which distributions produce the best results and to develop methods for achieving these ideal sampling distributions.

Acknowledgment

This research supported by NSF Grant #P20 RR16448.

References

Daida, Jason M., Tang, Ricky, Samples, Michael E., and Byom, Matthew J. (2006). Phase transitions in genetic programming search. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 1, pages –. Springer, Ann Arbor.

Daida, Jason M., Ward, David J., Hilss, Adam M., Long, Stephen L., Hodges, Mark R., and Kriesel, Jason T. (2004). Visualizing the loss of diversity in genetic programming. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1225–1232, Portland, Oregon. IEEE Press.

Dignum, Stephen and Poli, Riccardo (2007). Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In *GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1588–1595. ACM Press.

Dignum, Stephen and Poli, Riccardo (2008). Operator equalisation and bloat free GP. In O'Neill, Michael, Vanneschi, Leonardo, Gustafson, Steven, Esparcia Alcazar, Anna Isabel, De Falco, Ivanoe, Della Cioppa, Antonio, and Tarantino, Ernesto, editors, *Proceedings of the 11th European Conference on Genetic Programming*, volume 4971 of *LNCS*, pages 110–121, Naples. Springer.

Langdon, W. B. (1999). Size fair and homologous tree genetic programming crossovers. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1092–1097, Orlando, Florida, USA. Morgan Kaufmann.

Poli, Riccardo, Langdon, William B., and Dignum, Stephen (2007). On the limiting distribution of program sizes in tree-based genetic programming. In Ebner, Marc, O'Neill, Michael, Ekárt, Anikó, Vanneschi, Leonardo, and Esparcia-Alcázar, Anna Isabel, editors, *Proceedings of the 10th European*

Conference on Genetic Programming, volume 4445 of *Lecture Notes in Computer Science*, pages 193–204, Valencia, Spain. Springer.

Poli, Riccardo and McPhee, Nicholas Freitag (2001). Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In Miller, Julian F., Tomassini, Marco, Lanzi, Pier Luca, Ryan, Conor, Tettamanzi, Andrea G. B., and Langdon, William B., editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 126–142, Lake Como, Italy. Springer-Verlag.

Poli, Riccardo, McPhee, Nicholas Freitag, and Vanneschi, Leonardo (2008). The impact of population size on code growth in gp: Analysis and empirical validation. In O'Neill, Michael, Vanneschi, Leonardo, Gustafson, Steven, Alcazar, Anna Isabel Esparcia, Falco, Ivanoe De, Cioppa, Antonio Della, and Tarantino, Ernesto, editors, *Genetic And Evolutionary Computation Conference, GECCO 2008*. ACM Press.

Soule, Terence and Foster, James A. (1998). Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation*, pages 781–186, Anchorage, Alaska, USA. IEEE Press.

Chapter 7

ANALYSIS OF THE EFFECTS OF ELITISM ON BLOAT IN LINEAR AND TREE-BASED GENETIC PROGRAMMING

Riccardo Poli[1], Nicholas F. McPhee[2] and Leonardo Vanneschi[3]
[1]*Department of Computing and Electronic Systems, University of Essex, UK;* [2]*Division of Science and Mathematics, University of Minnesota, Morris, USA;* [3]*Dipartimento di Informatica, Sistemistica e Comunicazione, University of Milano-Bicocca, Milan, Italy.*

Abstract Elitism, a technique which consists of copying, unchanged, one or more of the most fit individuals from one generation to the next, is widely used in generational evolutionary algorithms, including Genetic Programming (GP). Elitism ensures that the best individuals discovered in a generation (and hence in the whole run) are not lost, and, perhaps even more importantly, are made available to new generations for possible further improvements. In a recent study on the evolution of robustness in GP the average size of best of run individuals was reported to grow more slowly in the presence of elitism. This is an important finding, but no explanation was provided for why this happened nor whether this was a general effect. In this paper we model elitism mathematically and explain how, in general, elitism modulates the dynamics of the mean program size of the population, including both its positive and negative effects on bloat. Experimental results with two GP systems and four different problems corroborate the theory.

Keywords: bloat, elitism, size evolution equation, theory, linear GP, tree-based GP

1. Introduction

Bloat is one of the most widely studied aspects of GP, and many GP implementations use elitism. However, to the best of our knowledge, no substantial investigation of their relationships and dependencies has ever been proposed.

This paper takes significant steps towards filling that gap. We provide a mathematical model of elitism and explain how, in general, elitism modulates the dynamics of the mean program size of a GP population, including its effects on bloat. We also present corroborating experimental results obtained from applying two separate GP systems to four different test problems. We find that

in fact elitism can have a significant impact, often substantially slowing the rate of bloat in the later generations of a run.

In the remainder of this section we look at what is known about both areas (bloat and elitism) and we survey the key the contributions of this paper. In Section 2 we present our theoretical analysis of the effects of elitism. We test our theory in Section 3, where we report empirical results. We then conclude in Section 4.

Bloat

It has long been known that in many conditions GP populations exhibit the phenomenon of *bloat*—a rapid increase in program size not accompanied by any significant corresponding increase in fitness. There has been considerable debate as to the causes of this phenomenon and several qualitative theories have been proposed. For example, the *replication accuracy theory* (McPhee and Miller, 1995) states that GP evolves towards (bloated) representations because this increases the replication accuracy of individuals. The *removal bias theory* (Soule and Foster, 1998) observes that inactive code in a GP tree (code that is not executed, or is executed but its output is then discarded) tends to be low in the tree, residing therefore in smaller-than-average-size subtrees. Crossover events excising inactive subtrees produce offspring with the same fitness as their parents. On average the inserted subtree is bigger than the excised one, so such offspring are bigger than average while retaining the fitness of their parent, leading ultimately to growth in the average program size. Another important theory, the *nature of program search spaces theory* (Langdon and Poli, 1997; Langdon et al., 1999), predicts that above a certain size, the distribution of fitnesses does not vary with size. Since there are more long programs, the number of long programs of a given fitness is greater than the number of short programs of the same fitness. Over time GP samples longer and longer programs simply because there are more of them. Finally, the *crossover bias theory* (Poli et al., 2007; Dignum and Poli, 2007) states that crossover pushes the population towards a particular distribution of program sizes, where small programs have a much higher frequency than longer ones. Because in most problems very small programs have low fitness, programs of above average length have a selective advantage over programs of below average length, and, so, the mean program size increases generation after generation.

In (Poli and McPhee, 2003), the following size evolution equation was derived which provides the expected average size of the programs at the next generation in a GP population under the effects of selection, reproduction, and many types of crossover (including standard sub-tree crossover):

$$E[\mu(t+1)] = \sum_l S(G_l)p(G_l,t). \tag{7.1}$$

Here $\mu(t)$ is the mean size of the programs in the population at generation t, $E[\,]$ is the expectation operator, G_l is the set of all programs of a particular shape (shapes are indexed by l), $S(G_l)$ is the size (number of nodes) of programs of shape l and $p(G_l, t)$ is the probability of selecting programs of shape l from the population in the current generation (indexed by t). This equation is a useful theoretical tool to understand program-size changes, and we will make use of this result in the next section to explain how elitism affects bloat. Additionally, it has been used in the design of bloat control techniques such as (Poli, 2003). However, in itself Equation 7.1 does not explain the reasons for bloat: it only describes what kind of things can happen size-wise in a GP run. That is, the size evolution equation effectively says that bloat can only be the result of an inbalance in the selection probabilities (ultimately fitnesses) for differerent length classes of programs. What it does not do is to say why such differences in selection probability exist. A theory of bloat must explain that.

Important efforts to provide quantitative explanations for bloat were also made in (Banzhaf and Langdon, 2002) and (Rosca, 2003) where representation-less executable models of GP were proposed. In these a population of individuals is evolved, where each individual consists of only three numerical values: the fitness, the size of active code and the size of inactive code. Individuals are manipulated by selection and simple abstractions of mutation (Banzhaf and Langdon, 2002) and crossover (Rosca, 2003). Various effects were observed that were similar to corresponding effects found in GP runs.

Despite these various efforts, it is still unclear how the theories we have briefly reviewed above are related, how they fit within the theoretical framework provided by Equation 7.1 or the executable models of (Banzhaf and Langdon, 2002) and (Rosca, 2003), whether there is a single cause of bloat (in which case, presumably, only one of the theories is correct) or whether there are multiple causes (in which case, perhaps, different theories capture different aspects of the bloat phenomenon).

Elitism

Elitism is a commonly used technique where one or more of the highest-fitness individuals are copied, unchanged, from one generation to the next. The amount of elitism used is often characterised by the *elite fraction*, which is the ratio N/M between the elite size, N, and the population size, M.

Elitism is typically used in generational EAs (including GP) to ensure that the best individuals discovered in a generation (and hence in the whole run) are not lost, and, perhaps even more importantly, are made available for possible further improvements in subsequent generations. In certain conditions (Rudolph, 1994), this property can even provably make an EA a global optimiser (given enough generations).

The use of elitism is very widespread in GP. Many implementations use elites of size 1, although many systems, such as Luke's ECJ (Luke et al., 2006), allow for elites of any size. Indeed, GP has successfully been run with much bigger elites. For example, elite fractions of 1% (Piszcz and Soule, 2006), 2.5% (Voss and Foley, 1999), 4% (Yanai and Iba, 2001), 5% (Sastry et al., 2004), 10% (Topchy and Punch, 2001) and even 20% (Alvarez et al., 2000) have been reported in the literature.

An interesting but not widely reported connection exists between generational systems with elitism and steady state GP systems, where new individuals are immediately inserted in the population as soon as they are created. When new individuals replace the worst individual in the population, a steady state system is equivalent to a generational system with an elite of size $N = M - 1$. So, the theoretical results derived in this paper also apply to this kind of steady state GP system.

Linking Elitism and Bloat

In a recent study of the evolution of robustness in GP, (Piszcz and Soule, 2006) used different degrees of elitism in one of their four sets of experiments (Experiment 3, a symbolic regression problem with $\sin(x)$ as target function). Their goal was to better understand how different levels of elitism affected the robustness of best-of-run individuals. They also noticed, however, that the average size of individuals increased more slowly as the elite fraction was increased from 0 to 1% in steps of 0.2%. A detailed study of elitism was not the objective of that paper, so there was no theoretical explanation for why this happened, or whether this was a general effect beyond the specific regression problem and the levels of elitism considered.

In this paper, we aim to better understand how and why elitism influences the evolution of program size in GP. We show that elitism has predictable and general effects on average program sizes. We model these effects mathematically, showing under which conditions (population size, elite size, selection pressure, etc.) elitism encourages or inhibits program size growth. Finally, we provide experimental results which corroborate the theory, using both a linear GP system and a tree-based GP system on four different problems.

2. Size evolution in the presence of elitism

Equation 7.1 was originally derived under the assumption that the GP system is a pure generational system and that only selection and crossover are acting on the population (Poli and McPhee, 2003). Below we look at how the equation changes if a proportion of the next generation is created by copying some of the individuals in the previous generation. Then we discuss some possible implications of the theory.

Theory

Let M be the population size, and let us assume that our first step in the creation of a new generation is copying N maximally fit individuals over from the previous generation. We will call these individuals the *elite*.

We define $p_e(G_l)$ to be the proportion of individuals of shape l in the elite. Note that the elite can be thought of as the result of applying a round of truncation selection (Thierens and Goldberg, 1994) with truncation ratio N/M to the population. So, the quantity $p_e(G_l, t)$ is effectively the selection probability for programs of shape l under truncation selection. Let us assume that we have no ties across the elite boundary, i.e., there is a unique set of N best individuals.[1] Then we have

$$p_e(G_l, t) = \frac{1}{N} \sum_{x \in G_l} \delta(f(x) \geq f_N(t)), \tag{7.2}$$

where $\delta(\ )$ is an indicator function which returns 1 if its argument is true and 0 otherwise, the sum is over all programs of shape G_l in the population at generation t, $f(x)$ is the fitness of program x and $f_N(t)$ is the fitness of the N-th best-fit individual in the population at generation t.

Let $\mu_1(t)$ denote the average size of the individuals at time t in the first N slots of the new population, i.e., those filled by the elite of the previous generation. Then $E[\mu_1(t+1)] = \sum_l S(G_l) p_e(G_l, t)$, where the summation is extended to all possible shapes. The remaining $M - N$ individuals in the new generations are created in the usual way via standard reproduction, selection and recombination. Although we use these operations to create, in effect, a smaller population (of size $M - N$ instead of M), the statistical features of the individuals in this smaller population, such as mean program size, should remain the same. Thus the average size of the programs in the new generation but not in the elite is $E[\mu_2(t+1)] = \sum_l S(G_l) p(G_l, t)$, as prescribed by Equation 7.1.

In expectation, we have $N \times E[\mu_1(t+1)]$ nodes in the elite and $(M-N) \times E[\mu_2(t+1)]$ nodes in the rest of the new population. The expected program size at the next generation in the presence of elitism, $E[\mu_e(t+1)]$, is then obtained by adding these two groups of nodes and dividing by the population size. That is

$$\begin{aligned} E[\mu_e(t+1)] &= \frac{N}{M} E[\mu_1(t+1)] + \frac{M-N}{M} E[\mu_2(t+1)] \\ &= \frac{N}{M} \sum_l S(G_l) \big(p_e(G_l, t) - p(G_l, t)\big) + \sum_l S(G_l) p(G_l, t). \end{aligned}$$

[1] The handling of ties can potentially have subtle effects on an evolutionary system. See page 98 for more.

Then, using Equation 7.1 we obtain

$$E[\mu_e(t+1)] = \frac{N}{M} \sum_l S(G_l)\big(p_e(G_l,t) - p(G_l,t)\big) + E[\mu(t+1)] \quad (7.3)$$

In other words, elitism modulates the expected mean program size at the next generation. In particular, it can contribute to increasing the mean program size if, on average, $p_e(G_l,t) > p(G_l,t)$ for the larger than average program shapes, or $p_e(G_l,t) < p(G_l,t)$ for the smaller than average program shapes or both. (The opposite happens if the inequalities are inverted.) Furthermore, everything else being equal, the effect is modulated by the elite fraction N/M.

With this theoretical result in hand, we are now in a position to try and understand more about the magnitude and the nature of the effects of elitism in different conditions and phases in a GP run.

Insights and Predictions

Theory is often evaluated on the basis of the number and quality of the insights and predictions it allows. Below we look at whether the effects of elitism on program size should be expected to be large or small. Then we consider what effects one should expect to see in the different phases of a run.

Large or Small Effects. As we noted in Section 1 many GP implementations use elites of size $N = 1$. Since GP is often run with large populations, the elite fraction, N/M, in Equation 7.3 is frequently small (e.g., 0.01–0.1%). So, one might expect to see a limited modulation of elitism on the mean program size evolution.

This may not necessarily be the case, however, because Equation 7.3 represents only the expected behaviour of the system over a single time step. Over multiple time steps, we should expect to see the effects of elitism amplified. Furthermore, elitist GP has successfully been run with small populations (e.g., see (Gathercole and Ross, 1997)) or even tiny populations (e.g., Miller's Cartesian GP (Miller, 1999)) where an elite of size $N = 1$ may actually correspond to an elite fraction of 1% to 25%. Small populations are also often associated with long runs, which we would expect to further amplify any effects of elitism on program size. Finally, as we indicated in Section 1, steady state GP is widely used and is likely to behave somewhat like a generational system with a very large elite size. In such cases, we should expect significant changes in GP's behaviour induced by elitism. Indeed, as we will see in Section 3, the impact of the elite size on size evolution is marked even for elite fractions as small as 1%.

Faster or Slower Growth. So far we have not said whether we expect $E[\mu_e(t+1)]$ to be bigger or smaller than $E[\mu(t+1)]$, i.e., whether elitism

induces faster or slower growth.[2] We cannot give a general answer, since this depends on the correlation between fitness and length present in each particular generation. Below, however, we consider two situations where we can use the theory to make reasonable, general predictions: the early stages of a run, and the late, stagnating stages of a run. Because the details depend so heavily on the particulars of the problem and the run, the arguments below are often fairly qualitative.

Early Stages. GP runs typically start with populations of relatively small programs. In particular, standard initialisation methods such as the ramped-half-and-half and the grow method, tend to produce large proportions of very small programs. As noted in (Dignum and Poli, 2007), solutions are rarely found among very short programs, at least for problems of practical interest. As a result, longer than average programs are often fitter than shorter than average ones in the early stages of a run. Selection will, therefore, promote the use of the longer programs, which in turn leads to a growth in mean program size. Note that this growth cannot really be considered "bloat" since it is often associated with rapid improvements in mean and best fitness (i.e., fitness and length are positively correlated).

What should we expect elitism to do in these conditions? It very much depends on the aggressiveness of the selection scheme used and the fitness diversity in the population.[3] If the better (and typically longer) than average programs only have a slightly higher probability of being selected than the others, then the introduction of elitism could substantially increase the speed at which the mean program size grows. This is because the fraction of the population created by elitism is effectively created via truncation selection, which is typically characterised by a high selection pressure. So, unless one is already using a very strong form of selection, or the initial population is already made up of large programs, elitism should make it easier for GP to move the search towards the longer, more fit programs. Furthermore, it is reasonable to expect that the larger the elite fraction the more pronounced this effect would be. Of course, there is a limit to this, since as the elite size, N, grows, the selection pressure exerted by truncation selection decreases (see Equation 7.3).

[2]Here we are assuming that some form of growth *is* present in our GP runs. The theory presented in Section 2 is valid whatever the size-evolution behaviour of GP. So, it is applicable also to those rare cases where one observes average program size reductions or where average program size remains more or less constant in GP runs. We omit the treatment of these cases here due to space limitations.

[3]Effectively all selection schemes are equivalent if there are no fitness differences between members of the population. Conversely, a relatively weak selection scheme, such as fitness proportionate selection, can become very aggressive if there are extreme fitness differences.

To reiterate, however, if the standard selection scheme used in a GP run already provides strong selection pressure at the beginning of a run,[4] then elitism may actually weaken the selection pressure, thereby slowing down growth in the early stages of a run, and doing so proportionally to the elite fraction. (In the next section we explain that this is also expected to happen in the late stages of a run but for different reasons.)

As we will see in Section 3, both of these beginning-of-run scenarios are encountered in real runs, with elitism increasing growth in most of the cases, but also slowing it down in particular conditions.

Late Stages. There are good reasons to believe that in the late stages of a run, when most of the search momentum has run out and fitness increases have become rare, elitism would slow down bloat, and that the effect would be increased as the elite fraction increases. To see this, let us consider the following argument.

Since the expected average size of the programs outside the elite is the same as the expected average size of the programs in the absence of elitism, i.e., $E[\mu_2(t+1)] = E[\mu(t+1)]$, whether $E[\mu_e(t)]$ is bigger or smaller than $E[\mu(t+1)]$ depends entirely on the expected mean size of the programs in the elite, namely $E[\mu_1(t+1)]$. The question then is, what is the average size of these programs?

This is not easily answered mathematically, but it is clear that, particularly during the late, stagnating phases of a run and when using small populations, not all members of the elite change in one generation. For example, the fittest individual in the elite will remain the same until a superior individual is found. As a consequence the programs in the elite will often come from many generations before the current one.

If bloat is acting on the population, programs generated in previous generations tend to be smaller on average than the programs generated in the current generation. Therefore, in these conditions, an aging elite will tend to have programs of an average size which is significantly smaller than the expected program size at the next generation. In other words, $E[\mu_1(t+1)] < E[\mu(t+1)]$ and, so, $E[\mu_e(t+1)] < E[\mu(t+1)]$. Because of the factor N/M in Equation 7.3, the effect should be modulated by the elite fraction, with larger elite showing a more pronounced reduction in the rates of bloat.

Elitism, Bloat and Sorting Algorithms. Elites are often computed by first sorting the population by fitness and then taking the first N individuals to form

[4]This can happen for a variety of reasons, for instance, because the selection scheme imposes a high selection pressure or because there are extreme fitness differences among the members of the population.

the elite.[5] The result of this sorting is not fully specified in the presence of ties, i.e., when there are individuals with identical fitness. When ties occur within a group of individuals that lay across the elite boundary, then the elite is not uniquely determined. The specific details of the sorting algorithm will dictate which individuals will in fact be copied into the elite.

In this situation, stable sorting algorithms[6] can (depending on other details of the system's implementation) tend to copy the same set of individuals into the elite from one generation to the next. Therefore, successful individuals will tend to persist longer in the elite, thereby increasing the difference between the average size of the individuals in the elite and the average size of the individuals in the rest of the population. If, instead, the sorting algorithm randomises ties, then the average age of the programs in the elite may be less and their sizes correspondingly greater.

That different sorting algorithms may produce different elitist behaviours is important, since there may be conditions under which the effects of the specific choice of sorting algorithm could become significant. We expect, however, these effects to be only visible in small and discrete fitness domains, in particularly flat or "needly" landscapes, or in landscapes with extensive neutrality, where ties are more probable. In the experiments reported below we used *quicksort* (which is not typically stable) and *bubble sort* (which in our case was stable). We did not see substantial differences in behaviour across different domains and algorithms, suggesting that typically any differences are small and merely quantitative, rather than qualitative.

3. Experimental Results

We used two GP systems and two test problems for each. We describe these in the next sub-section. Then we describe the size-evolution behaviours recorded in our experiments with different degrees of elitism. Finally, we briefly discuss the impact of elitism on GP's problem-solving effectiveness.

GP Systems, Problems and Primitive Sets

Linear GP. The first system we used is a linear generational GP system. It initialises the population by repeatedly creating random individuals with lengths uniformly distributed between 1 and 100 primitives. The primitives are drawn randomly and uniformly from each problem's primitive set. The

[5] Obviously when N is fixed and very small (e.g, $N = 1$ or $N = 2$), it is faster to identify the elite by performing a linear scan of the population during which one stores the best N individuals found so far. For larger values of N, however, sorting is more efficient.

[6] Stable sorting algorithms maintain the relative order of items with equal comparison values, e.g., individuals with equal fitness.

system uses fitness proportionate selection and crossover applied with a rate of 100%. Crossover creates offspring by selecting two random crossover points, one in each parent, taking the first part of the first parent and the second part of the second w.r.t. their crossover points. We used populations of size 100 and 1000. For populations of size 100 we performed 500 independent runs. For populations of size 1000 we performed 100 independent runs. Runs lasted 100 generations.

With the linear GP system we used two families of test problems: `Polynomial` and `Lawn-Mower`. `Polynomial` is a symbolic regression problem where the objective is to evolve a function which fits a degree d polynomial of the form $x + x^2 + \cdots + x^d$, for x in the range $[-1, 1]$. Polynomials of this type have been widely used as benchmark problems in the GP literature. In particular we considered degree $d = 6$, and we sampled the polynomial at the 21 equally spaced points $x \in \{-1.0, -0.9, \ldots, 0.9, 1.0\}$. We call the resulting problem instance `Poly-6`. Fitness (to be minimised) was the sum of the absolute differences between target polynomial and the output produced by the program under evaluation over these 21 fitness cases. For this problem we considered a primitive set including the following instructions: `R1 = RIN`, `R2 = RIN`, `R1 = R1 + R2`, `R2 = R1 + R2`, `R1 = R1 * R2`, `R2 = R1 * R2`, and `Swap R1 R2`. The instructions refer to three registers: the input register `RIN` which is loaded with the value of x before a fitness case is evaluated and the two registers `R1` and `R2` which can be used for numerical calculations. `R1` and `R2` are initialised to x and 0, respectively. The output of the program is read from `R1` at the end of its execution.

`Lawn-Mower` is a variant of the classical Lawn Mower problem introduced by Koza in (Koza, 1994). As in the original version of the problem, we are given a square lawn made up of grass tiles. In particular, we considered lawns of size 10×10. The objective is to evolve a program which allows a robotic lawnmower to mow all the grass. In our version of the problem (which differs slightly from Koza's), a robot can perform one of three actions at each time step: move forward one step and mow the tile it lands on (`Mow`), turn left by 90 degrees (`Left`) or turn right by 90 degrees (`Right`). Fitness (to be minimised) was measured by the number of tiles left unmowed at the end of the execution of a program. We limited the number of instructions allowed in a program to a small multiple of the number of tiles available in the lawn (400 in these experiments).

Tree-based GP. For these experiments we adapted Fraser and Weinbrenner's tree-based GPC++ (Fraser and Weinbrenner, 1997) so as to allow the use of elites of any size. We considered two classical problems: the `Ant` problem and the `Even-5 Parity` problem. In both cases we used the standard primitives as described in (Koza, 1992), using populations of size 100 and 1000. Runs lasted

300 generations for `Ant` and 500 generations for `Even-5 Parity`. We used fitness proportional selection and Koza's subtree crossover applied with either 80% or 100% probability. In all conditions we performed 100 independent runs.

Size Evolution Results

We ran all the configurations outlined in the previous section using six different elite sizes: 0%, 1%, 5%, 10%, 30%, and 50% of the population. In this paper we will only report the results with the larger populations (size 1000) due to space limitations. However, the results with populations of size 100 were qualitatively similar.

Figure 7-1 shows the results we obtained with our linear GP system on the Poly-6 symbolic regression problem for the different elite percentages. The main thing to note here is that, while all systems behaved rather similarly in the early generations, elitism seems to induce a reduction in the rate of bloat in the later generations, with the runs using the bigger elites bloating more slowly than those with smaller elites and than "no elite" case. By generation 100, the runs with elite fractions $\leq 10\%$ had average sizes that were nearly twice as large as those using 50% elitism, for example. In the early generations we don't see a quicker growth in the presence of elitism (as was suggested as a possibility in Section 2) because in symbolic regression problems of the kind considered here, fitness values across the population have very high variance in the early generations, making fitness proportional selection more aggressive than truncation selection. Essentially the same can be seen for the `Lawn-Mower` problem, as illustrated in Figure 7-2, where initially there is a strong correlation between length and fitness.

For tree-based GP, however, we see the two phases postulated in Section 2, both in the `Ant` problem (Figures 7-3 and 7-5) and in the `Even-5 Parity` problem (Figures 7-4 and 7-6). In all cases we see that, initially, the stronger the elitism the faster the growth in program size. The picture is, however, reversed when the population settles into a stagnating phase. There, as predicted, the rate of bloat is very markedly reduced by increasing the elitism in the system, with elite fractions of 30% and 50% essentially behaving almost identically. This is consistent with our prediction that there would be a limit to the effects of additional elitism. Note also that elitism affects bloat in essentially the same way whether or not the reproduction operator is used to create a proportion of the individuals in the new generation as one can see by comparing Figure 7-3 with Figure 7-5 and Figure 7-4 with Figure 7-6.

Only one case *appears* to deviate from what we expected. This is represented by the runs of the `Even-5 Parity` problem where no elitism was used. These runs (see Figures 7-4 and 7-6) appear not to bloat at all, contrary to our

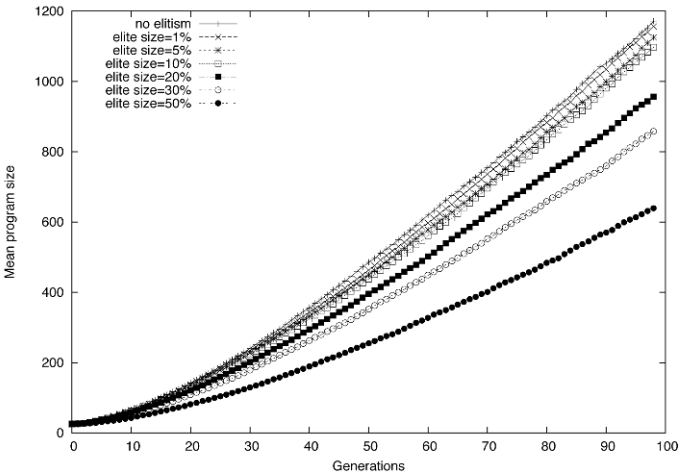


Figure 7-1. Plots of the average size of programs vs generation in a linear GP system with populations of size 1000 solving the Poly-6 problem for different elite fractions. Plots are averages over 100 independent runs.

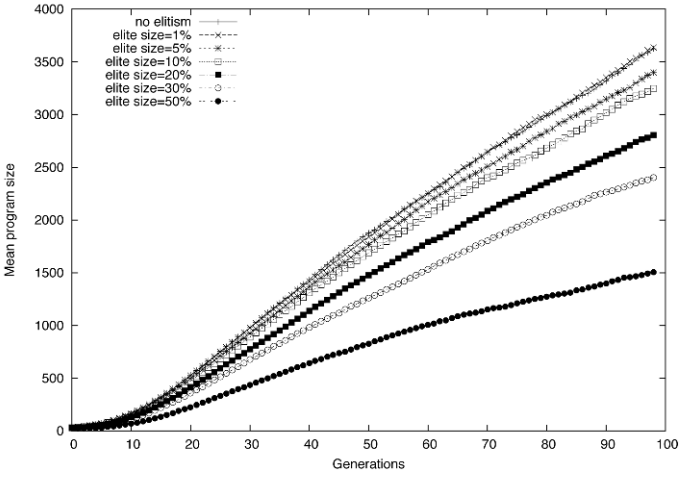


Figure 7-2. Plots of the average size of programs vs generation in a linear GP system with populations of size 1000 solving the lawnmower problem for different elite fractions. Plots are averages over 100 independent runs.

expectation that the no-elitism case would be, in general, the fastest growing. The reason for this is very simple. In this problem, with a population of 1,000 individuals, most (if not all) programs in the first generation satisfy 16 out of the 32 fitness cases (see for instance (Langdon and Poli, 2002; Vanneschi, 2004; Vanneschi et al., 2006)). When an improvement is found, it typically is a program that satisfies one extra case. With fitness proportionate selection this

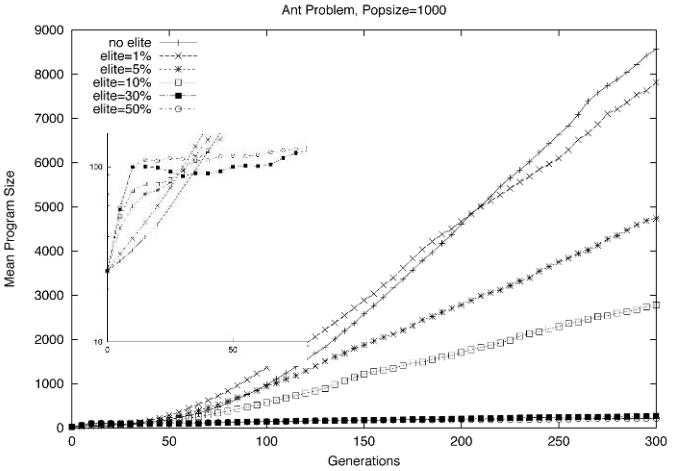


Figure 7-3. Plots of the average size of programs vs generation in a tree-based GP system with populations of size 1000 solving the `Ant` problem for different elite fractions and a crossover rate of 100%. Plots are averages over 100 independent runs. The first 50 generations are also shown in the inset with a log scale. Note that the values for 30% elite and 50% elite were almost identical, and the 30% values tend to obscure the 50% values in this plot.

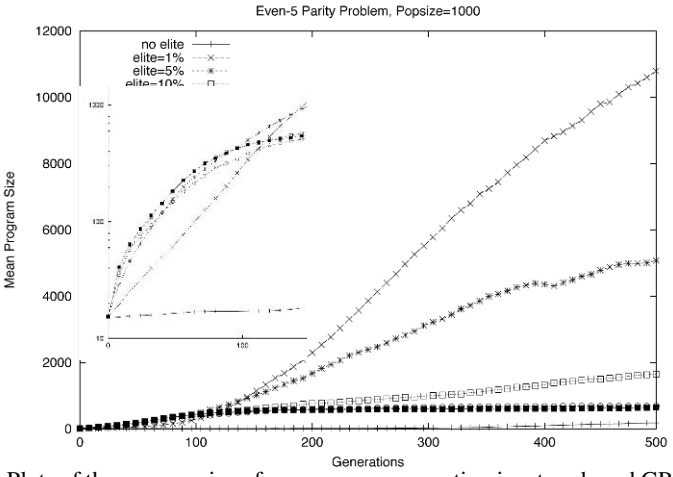


Figure 7-4. Plots of the average size of programs vs generation in a tree-based GP system with populations of size 1000 solving the `Even-5 Parity` problem for different elite fractions and a crossover rate of 100%. Plots are averages over 100 independent runs. The first 200 generations are also shown in the inset with a log scale. Note that the values for 30% elite and 50% elite were almost identical, and the 30% values tend to obscure the 50% values in this plot.

produces a very small increase in the selection probability for such a program. Since we use 100% crossover, without elitism this improved program is very likely to be destroyed at the next generation. Its offspring (which statistically might be slightly longer then average) are likely to have the same fitness as the

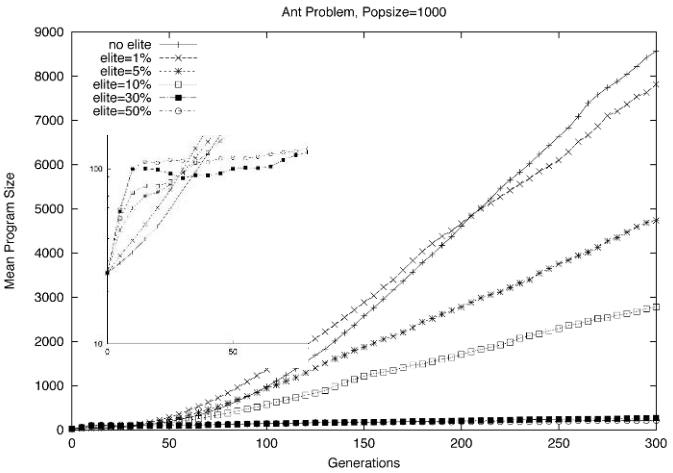


Figure 7-5. Plots of the average size of programs vs generation in a tree-based GP system with populations of size 1000 solving the `Ant` problem for different elite fractions and a crossover rate of 80%. Plots are averages over 100 independent runs. The first 50 generations are also shown in the inset with a log scale.

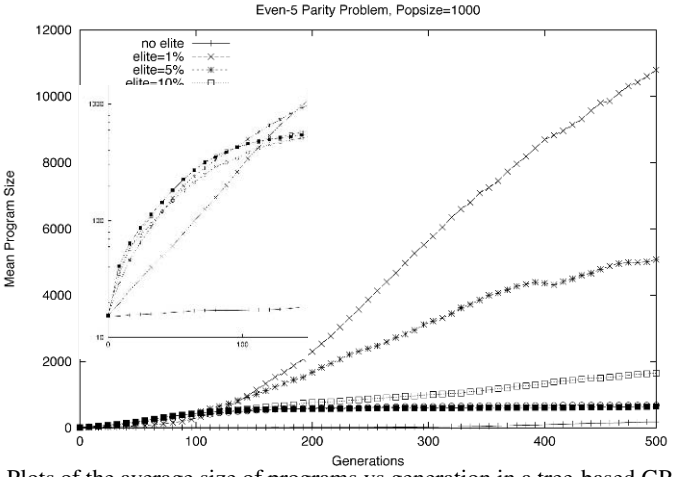


Figure 7-6. Plots of the average size of programs vs generation in a tree-based GP system with populations of size 1000 solving the `Even-5 Parity` problem for different elite fractions and a crossover rate of 80%. Plots are averages over 100 independent runs. The first 200 generations are also shown in the inset with a log scale.

rest of the population. So, they are not likely to be reselected. In short, there is very little correlation between size and fitness, so many generations are needed before significant progress can be made on the problem and bloat can really set in. Eventually it does so; notice the small rise in the very bottom right corner of the figure, which corresponds to an average size of 67.1 nodes. For populations of 100, the final average size was 160.9. In both cases the average size in the

initial population was 15.3. We conjecture that if given many more generations populations without elitism would eventually overtake those with elitism.

Fitness and Success Rates

In order to convey a fuller picture of the effects of elitism, we include some information on fitness and success rates for the various combinations studied here.

That the `Even-5 Parity` problem cannot be solved with populations of 1000 individuals without elitism is very apparent in the fitness plot shown in Figure 7-7 (bottom). In the absence of elitism the mean fitness of the population (not shown) improves extremely slowly from its initial value of 16. Furthermore, the average over the 100 runs of the best fitness in the population (which, without elitism, cannot be retained) does not improve from its initial value of around 14 and, in fact, slightly worsens over time (Figure 7-7, bottom). As soon as elitism is added, however, there is something like a phase transition, with the larger elite fractions leading to larger advantages both in terms of mean fitness and best fitness. This is consistent with (Vanneschi et al., 2006) where this problem was shown to present a large neutral network with fitness 16 from which one can reach individuals of high fitness only by following very narrow paths. Each path is formed by a chain of neighbouring individuals that are one mutation apart. Thus, the only way to escape from the network is to ensure individuals of fitness higher than 16 are kept in the population and progressively further improved. This is why, effectively, in the parity problem elitism not only provides smaller solutions, but it also makes the search for solutions easier.

Figure 7-7 also shows the best fitness plots for the `Ant` problem with different levels of elitism. Because of the nature of the problem and the higher granularity of the fitness function, here the absence of elitism does not produce catastrophic results. Elitism, however, does appear to be advantageous up to a point both in terms of mean population fitness and in terms of best fitness in each generation. Best results are obtained with small amounts of elitism, e.g., 1%.[7] Increasing the elite fraction slightly decreases performance. So, for the `Ant` problem one can trade accuracy for solution size by changing the elite size. In this sense, elitism acts as a typical anti-bloat method.

A similar kind of behaviour was also observed with the linear GP system when solving the `Poly-6` problem. As shown in Table 7-1, the success rate tended to decrease slightly at the highest elite fractions (50%, and possibly also 30%), although it its not clear whether there is any significant difference with smaller elites. The table also shows the mean best fitness (error) obtained

[7]This does not mean that the traditional $N = 1$ is best. The smallest value of the elite fraction in our runs was 1%, which corresponds, for populations of 1000, to an elite of $N = 10$ individuals.

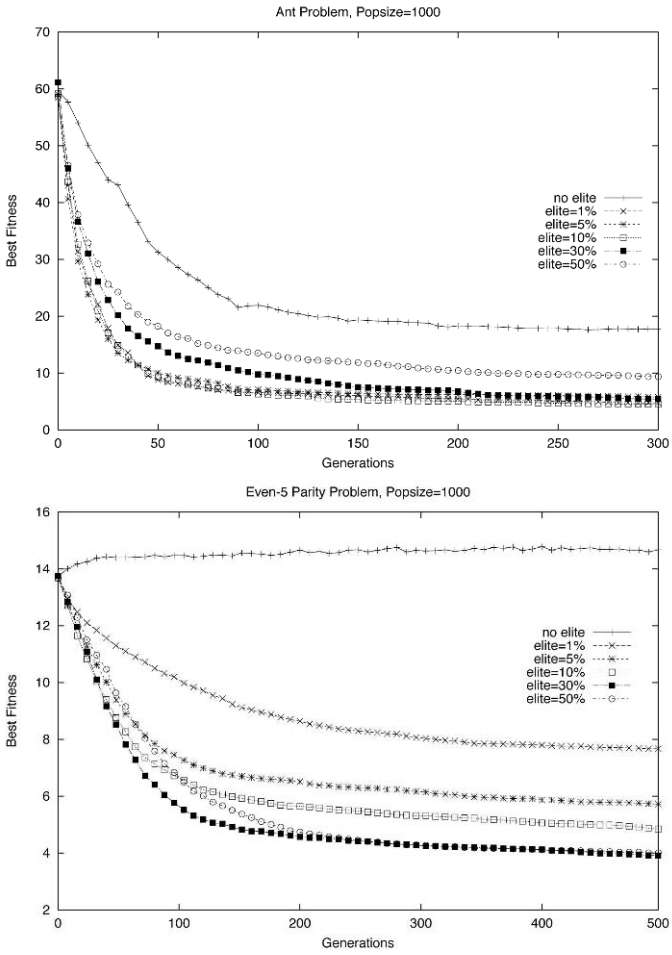


Figure 7-7. Plots of the best fitness of programs vs generation in a tree-based GP system with populations of size 1000 solving the ANT (top) and Even-5 Parity (bottom) problems for different elite fractions and a crossover rate of 100%. Plots are averages over 100 independent runs.

in different configurations. This is substantially unaffected by the elite size, except, possibly, at the highest elite fraction. The success rates and fitnesses for the (easier) `Lawn-Mower` problem (not reported) showed no variation with the elite size.

4. Conclusions

Elitism is widely used in GP, particularly when the population is not very large and there is a greater chance of losing the best individual. Surprisingly, despite its widespread use, to the best of our knowledge there is very little

Table 7-1. Success rate and mean best fitness in runs with a linear GP system solving the poly-6 problem for different degrees of elitism. Figures were estimated based on 500 independent runs for population size 100 and 100 independent runs for population size 1000.

Population Size	*Elite Fraction*	*Success Rate*	*Mean Best Fitness*
100	No Elite	0.41	0.64
100	1%	0.34	0.60
100	5%	0.39	0.63
100	10%	0.38	0.62
100	30%	0.31	0.57
100	50%	0.24	0.51
1000	No Elite	0.71	0.85
1000	1%	0.76	0.88
1000	5%	0.65	0.82
1000	10%	0.78	0.89
1000	30%	0.69	0.84
1000	50%	0.53	0.75

literature and no theory on the effects of elitism in GP. The only case where different degrees of elitism were tested was one set of experiments (out of four) with one problem in the work by Piszcz and Soule (Piszcz and Soule, 2006), and the focus there was on the evolution of robustness in GP rather than on explaining the effects of elitism on bloat.

In this paper we provided a general mathematical model of the effects of elitism on program size evolution, a series of predictions on the qualitative behaviour of GP based on this model and a plethora of experimental results corroborating both the model and its predictions.

Our results make it clear that elitism can have a powerful effect on bloat, generally reducing the level of bloat in the later generations, with larger elite sizes typically controlling bloat more strongly. Elitism also improved performance in most cases. So, in general the use of elitism (possibly with bigger elites than the single individual used routinely) brings significant advantages to GP.

References

Alvarez, Luis F., Toropov, Vassili V., Hughes, David C., and Ashour, Ashraf F. (2000). Approximation model building using genetic programming methodology: applications. In Baranger, Thouraya and van Keulen, Fred, editors, *Second ISSMO/AIAA Internet Conference on Approximations and Fast Reanalysis in Engineering Optimization*.

Banzhaf, W. and Langdon, W. B. (2002). Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91.

Dignum, S. and Poli, R. (2007). Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In et al., Dirk Thierens, editor, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1588–1595, London. ACM Press.

Fraser, Adam and Weinbrenner, Thomas (1993, 1994, 1997). Gpc++ genetic programming c++class library.

Gathercole, C. and Ross, P. (1997). Small populations over many generations can beat large populations over few generations in genetic programming. In John R. Koza *et al.*, editor, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 111–118, San Francisco, CA, USA. Morgan Kaufmann.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, John R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

Langdon, W. B. and Poli, R. (1997). Fitness causes bloat. In Chawdhry, P. K., Roy, R., and Pant, R. K., editors, *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag London.

Langdon, W. B. and Poli, R. (2002). *Foundations of Genetic Programming*. Springer, Berlin, Heidelberg, New York, Berlin.

Langdon, William B., Soule, Terry, Poli, Riccardo, and Foster, James A. (1999). The evolution of size and shape. In Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter J., editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA.

Luke, S., Panait, L., Balan, G., Paus, S., Skolicki, Z., Bassett, J., Hubley, R., and Chircop, A. (2006). ECJ, a Java-based Evolutionary Computation Research System. Downloadable versions and documentation can be found at the following url: http://cs.gmu.edu/ eclab/projects/ecj/.

McPhee, Nicholas Freitag and Miller, Justin Darwin (1995). Accurate replication in genetic programming. In Eshelman, L., editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA. Morgan Kaufmann.

Miller, Julian F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142, Orlando, Florida, USA. Morgan Kaufmann.

Piszcz, Alan and Soule, Terence (2006). Dynamics of evolutionary robustness. In Keijzer *et al.*, M., editor, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 871–878, Seattle, Washington, USA. ACM Press.

Poli, Riccardo (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In Ryan, Conor, Soule, Terence, Keijzer, Maarten, Tsang, Edward, Poli, Riccardo, and Costa, Ernesto, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 204–217, Essex. Springer-Verlag.

Poli, Riccardo, Langdon, William B., and Dignum, Stephen (2007). On the limiting distribution of program sizes in tree-based genetic programming. In Ebner *et al.*, M., editor, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 193–204, Valencia, Spain. Springer.

Poli, Riccardo and McPhee, Nicholas Freitag (2003). General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2):169–206.

Rosca, Justinian (2003). A probabilistic model of size drift. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practice*, chapter 8, pages 119–136. Kluwer.

Rudolph, Günter (1994). Convergence analysis of canonical genetic algorithm. *IEEE Transactions on Neural Networks*, 5(1):96–101.

Sastry, Kumara, O'Reilly, Una-May, and Goldberg, David E. (2004). Population sizing for genetic programming based on decision making. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 4, pages 49–65. Springer, Ann Arbor.

Soule, Terence and Foster, James A. (1998). Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation*, pages 781–186, Anchorage, Alaska, USA. IEEE Press.

Thierens, Dirk and Goldberg, David (1994). Convergence models of genetic algorithm selection schemes. In Davidor, Yuval, Schwefel, Hans-Paul, and Manner, Reinhard, editors, *Parallel Problem Solving from Nature – PPSN III*, number 866 in Lecture Notes in Computer Science, pages 119–129, Jerusalem. Springer-Verlag.

Topchy, A. and Punch, W. F. (2001). Faster genetic programming based on local gradient search of numeric leaf values. In Spector, L., *et al.*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO '01*, pages 155–162, San Francisco, CA. Morgan Kaufmann.

Vanneschi, L. (2004). *Theory and Practice for Efficient Genetic Programming*. Ph.D. thesis, Faculty of Sciences, University of Lausanne, Switzerland.

Vanneschi, L., Pirola, Y., Collard, P., Tomassini, M., Verel, S., and Mauri, G. (2006). A quantitative study of neutrality in GP boolean landscapes. In M. Keijzer *et al.*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'06*, volume 1, pages 895–902. ACM Press.

Voss, Mark S. and Foley, Christopher M. (1999). Evolutionary algorithm for structural optimization. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 678–685, Orlando, Florida, USA. Morgan Kaufmann.

Yanai, Kohsuke and Iba, Hitoshi (2001). Multi-agent robot learning by means of genetic programming : Solving an escape problem. In Liu, Yong, Tanaka, Kiyoshi, Iwata, Masaya, Higuchi, Tetsuya, and Yasunaga, Moritoshi, editors, *Evolvable Systems: From Biology to Hardware: 4th International Conference, ICES 2001*, volume 2210 of *LNCS*, pages 192–203, Tokyo, Japan. Springer-Verlag.

Chapter 8

AUTOMATED EXTRACTION OF EXPERT DOMAIN KNOWLEDGE FROM GENETIC PROGRAMMING SYNTHESIS RESULTS

Trent McConaghy[1,2], Pieter Palmers[1], Georges Gielen[1] and Michiel Steyaert[1]

[1]*Katholieke Universiteit Leuven, Leuven, Belgium*
[2]*Solido Design Automation Inc., Saskatoon, Canada*

Abstract Recent work in genetic programming (GP) shows how expert domain knowledge can be *input* to a GP synthesis system, to speed it up by orders of magnitude and give trustworthy results. On the flip side, this chapter shows how expert domain knowledge can be *output* from the results of a synthesis run, in forms that are immediately recognizable and transferable for problem domain experts. Specifically, using the application of analog circuit design, this chapter presents a methodology to automatically generate a decision tree for navigating from performance specifications to topology (structure) choice; global nonlinear sensitivity analysis on the effect of structure and parameters to performance; and a means to extract analytical expressions of performance tradeoffs. The extraction uses a combination of data mining and GP. Results are shown for operational amplifier circuit design on a database containing thousands of Pareto Optimal designs across five objectives.

Keywords: synthesis, domain knowledge, knowledge extraction, insight, multi-objective, data mining, analog, integrated circuits, age layered population structure

1. Introduction

Engineers in many fields, from circuit design to automotive design, use their experience and intuition to choose design structures (topologies) and to design new structures to meet design goals. Unfortunately, the structure used may not be optimal, leading to suboptimal final design performances, cost, and robustness. The suboptimal design may occur because the design is using new or unfamiliar materials, the designer is too time-constrained to be thorough (via

time-to-market pressures), or simply because the designer just doesn't have the experience level to know what might be best. This last point is understandable for many disciplines; e.g. it is well recognized that gaining depth in analog circuit design is a process that takes years to get started and decades to master (Williams, 1991). All said, it still means that a suboptimal design structure may be used.

Hence, it is desirable to provide support for the designer in selection and design of such structures, and ideally to catalyze the designer's learning process. Prior research in automated analog circuit design tools such as (Koza et al., 2003) has focused on automated selection or synthesis of structures, but has had little emphasis on giving insight back to the user. This is a problem for two reasons: (1) By deferring control to automated tools, a designer's learning might slow, not to mention being less-equipped to cope when problems arise that the tools cannot handle, and (2) highly experienced designers can be reluctant to use any automated synthesis tools, because past tools have fallen far short of expectations while the designers have had much success doing their work manually.

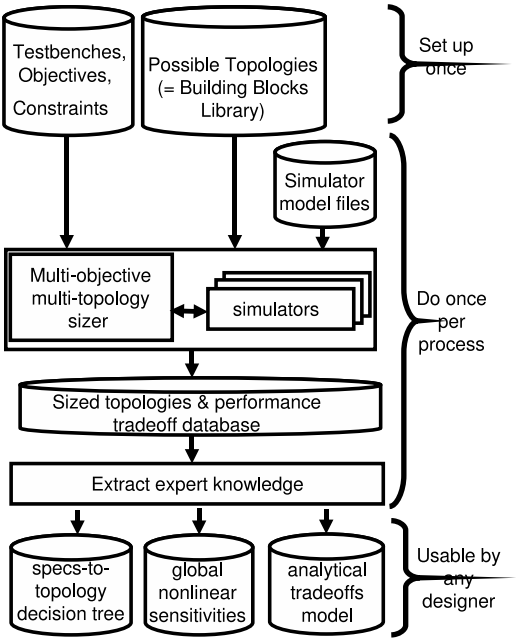


Figure 8-1. Target flow. A multi-objective multi-topology sizer generates a database, which is then data-mined to discover relations among design structure (topology), design parameters, and performances.

This chapter asks: is there a means to help designers maintain and build expert topology-performance knowledge, in such a fashion that even reluctant design-

ers might benefit? The starting point is MOJITO (McConaghy et al., 2007), synthesis system which traverses thousands of *trustworthy* circuit topologies using genetic programming (Koza, 1992), to automatically generate a database of Pareto-optimal sized topologies. This chapter's contributions are:

- A data-mining perspective on the Pareto-optimal database to extract the following expert knowledge: (a) a specifications-to-topology decision tree, (b) global nonlinear sensitivities on topology and sizing variables, and (c) analytical performance-tradeoff models.
- A suggested flow in which even reluctant users can conveniently use the extracted knowledge (Figure 8-1). The database generation and knowledge extraction only needs to be done once per semiconductor process node (i.e. every year or two), e.g. by a single designer or a modeling group. The extracted knowledge can be stored in a document (e.g. pdf, html), and simply made available to other designers.

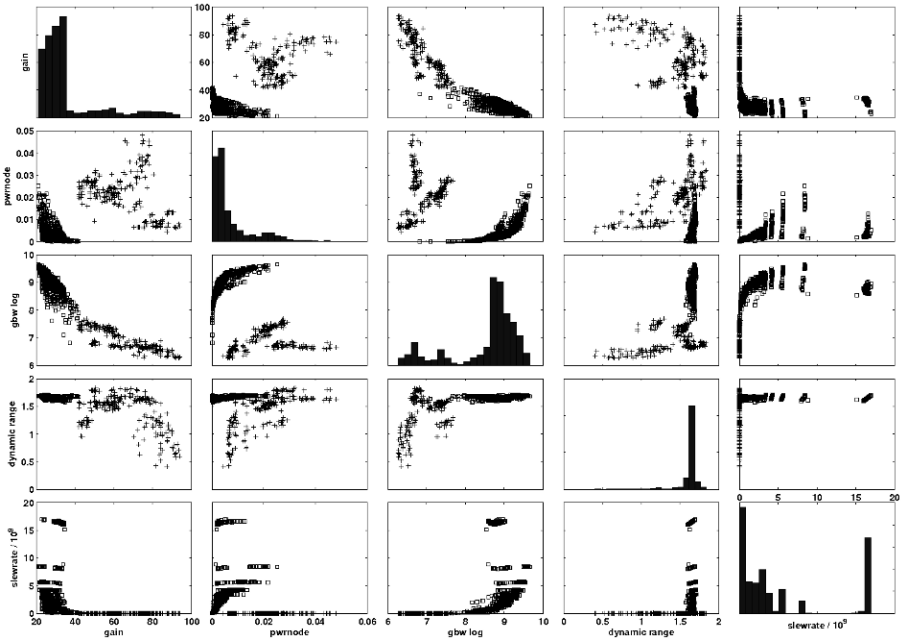


Figure 8-2. Grid illustrating the Pareto Front. The diagonal entries show histograms of performance; the rest show two-dimensional projections from the five objectives. The squares are 1-stage amplifiers, and the pluses are two-stage amplifiers.

This chapter's knowledge extraction procedures will be explained using a reference database, generated as described in section 2. Section 3 describes how a specifications-to-decision tree is extracted from the database. Section 4

describes extraction of global nonlinear sensitivities, and Section 5 extraction of analytical tradeoffs model. Section 6 concludes.

2. Generation of Database

This section describes the system used to generate the sized-topologies database, and the experimental setup.

We use the analog structural synthesis system MOJITO, which was described in (McConaghy et al., 2007). MOJITO has three main elements:

- It defines the space of possible structures as *hierarchically composed, pre-specified building blocks* . It starts at atomic blocks (e.g. transistors and resistors), and builds up through more complex blocks (e.g. current mirrors and differential pairs) all the way to the target block type (e.g. operational amplifiers). Each block can be viewed as a grammar symbol: the atomic blocks are terminals, and higher-level blocks are combined symbols and derivation rules. The grammar is parameterized: each symbol can have arguments which can control what lower-level derivations are taken or atomic blocks' parameters (e.g. transistor width, and resistance). An example library of (McConaghy et al., 2007) 30 building blocks combines into about 3000 different possible overall structures (topologies) with up to dozens of possible parameters per topology.
- MOJITO uses hierarchy-aware search operators in order to naturally mix sub-blocks, but also vector-aware search ("optimization") operators to refine atomic parts' widths, resistances, etc.
- The search algorithm is an evolutionary algorithm (EA), which uses an age-layered population structure (ALPS) (Hornby, 2006) to avoid premature convergence, and NSGA-II (Deb et al., 2002) for handling multiple objectives and constraints.

With this approach MOJITO overcomes the tissues of past analog structural synthesis systems such as (Koza et al., 2003), which had very high computational cost and gave non-trustworthy results. (That is, results were not htrustworthy enough to spend millions of dollars to fabricate the design (McConaghy and Gielen, 2005).)

We use MOJITO to search this space of sized topologies. The problem has five performance objectives: maximize GBW, minimize power, maximize gain, maximize dynamic range, maximize slew rate. A single synthesis run was done, using the same parameters as (McConaghy et al., 2007), except $N_{ind,max}$ = 200,000. Synthesis was run overnight on a Linux cluster having 30 cores of 2.5 GHz each (which is palatable for an industrial setting). 180 generations were covered, traversing 3528 possible topologies and their associated sizings. It returned a database of 1576 Pareto-Optimal sized topologies.

To become oriented with the raw results data, Figure 8-2 shows a grid of 2D scatterplots and histograms for the five performance objectives. From the histograms, we can get a quick picture of the distribution and bounds of performances. From the scatterplots, we begin to understand correlations and trends. Note how the one-stage topologies only occupy a different region of performance space and follow a markedly different performance trend than the two-stage topologies. The two-stage topologies have several sub-clusters of performances, hinting at further decomposition of topology types.

3. Extraction of Specifications-To-Topology Decision Tree

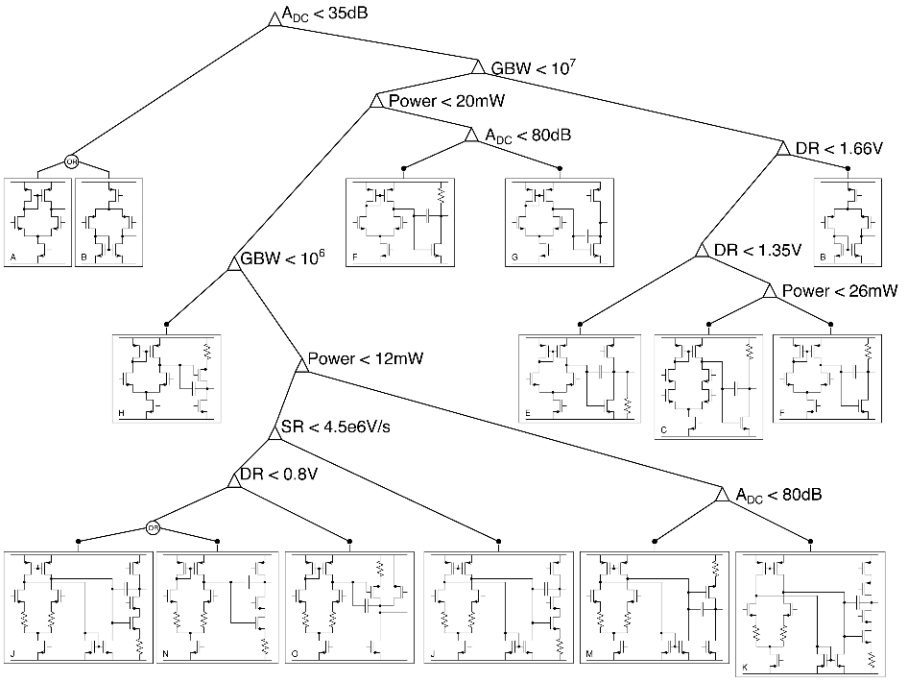


Figure 8-3. A decision tree for going from specifications to topology. Unlike past approaches, this was automatically generated. Technology is 0.18 μm CMOS, V_{dd} = 1.8 V.

.

Introduction

This section describes the automatic extraction of decision (CART) trees (Breiman et al., 1984) that map from performance values to topology choice. Decision trees have a double use: they can directly suggest a choice based on inputs, and also expose the series of steps underlying the decision. CART

trees are in widespread use, such as medicine: "In medical decision making (classification, diagnosing, etc.) there are many situations where decision must be made effectively and reliably. ... Decision trees are a reliable and effective decision making technique that provide high classification accuracy with a simple representation of gathered knowledge." Decision trees have not gone unnoticed in analog CAD either, as they have been proposed as the centerpiece of topology-choosing "expert systems", e.g. (Harjani et al., 1989). Unfortunately, these trees had to be manually constructed which took weeks to months of effort, and were based on rules of thumb that became obsolete as soon as the process node changed. In contrast, this chapter constructs the specifications-to-topology decision tree *automatically* from data. This is only possible now, because a prerequisite to get the data was a competent multi-topology multi-objective sizer that could output a diverse set of topologies, as MOJITO does.

Problem Formulation and Approach

We formulate specifications-to-topology decision tree induction as a classification problem from a Pareto Optimal Set $Z = \{\phi_1^*, \phi_2^*, \ldots, \phi_j^*, \ldots, \phi_{N_Z}^*\}$ resulting from a MOJITO run. Within Z, there are N_T unique topologies ($N_T \leq N_Z$) with corresponding class labels $\Upsilon = \{1, 2, \ldots, N_T\}$. For individual ϕ_j^*, let υ_j be its topology class label; $\upsilon_j \in \Upsilon$. Let $\boldsymbol{f_j}$ be the objective function values corresponding to υ_j: $\boldsymbol{f_j} = \{f_1(\phi_j^*), f_2(\phi_j^*), \ldots, f_{N_f}(\phi_j^*)\}$, an N_f – dimensional vector. Tree induction constructs a classifier ω that maps from $\boldsymbol{f_j}$ to υ_j, i.e. $\widehat{\upsilon_j} = \omega(\boldsymbol{f_j})$. ω can be viewed as a collection of N_R disjoint rectangular regions R_i, $i = 1..N_R$; where each region R_i has an associated class $\upsilon_i \in \Upsilon$.

Tree construction using the CART algorithm finds a tree ω in the space of possible trees Ω using a greedy algorithm. It begins with just a root node holding all data points $\{\boldsymbol{f_j}, \upsilon_j\}, j = 1..N_Z$ and therefore is represented by a single region R_1 covering all of input $\boldsymbol{f}$ space. Each objective i is a possible split variable, and the values $f_{i,j}$ for that objective comprise the possible split values (with duplicates removed). From among all possible $\{split\_variable, split\_value\}$ tuples in the data, the algorithm chooses the tuple with the highest information gain according to the chosen split criterion. That split creates a left and right child, where left child is assigned data points and region meeting $split\_variable \leq split\_value$, and the right child is assigned the remaining data points and region. The algorithm recurses, splitting each leaf node until a leaf node has too few points to split further. The final set of regions is defined by the leaf nodes' regions only. The tree-constructing implementation was in Matlab's "classregtree" routine. The "gini" splitting criterion was used; it selects the $\{variable, value\}$ that splits off the most data points (Breiman et al.,

1984). The minimum number of points per leaf node was 10 so that a compact tree would be generated.

Results and Discussion

Figure 8-3 shows the tree that was automatically generated. It provides insight into what topologies are appropriate for which performance ranges, and actually even gives a suggested topology from a set of input specifications.

We see that the first decision variable, at the top of the tree, is low-frequency gain (A_{DC}). Following through the tree, we see that all specifications (objectives) play a role for selecting some topologies: gain-bandwidth (GBW), power, slew rate (SR), and dynamic range (DR). When the specifications require low gain, the tree suggests single-stage topologies (the simplest topologies, A and B); and two-stage topologies when a higher gain is required (the remaining topologies). To a circuit designer, this makes perfect sense. There is further refinement on the type of topology: in cases where a very large gain is required with a limited power budget, a two-stage amplifier with large degrees of "cascoding" and "source degeneration" is suggested (the bottom row of topologies J, K, M, N, O). "Cascoding" is where there are two transistors arranged vertically in series; "source degeneration" is where there are resistors in the left half of a topology. If power is less of an issue, one can also use a non-cascoded two-stage amplifier (topologies F and G). Since only Pareto-optimal individuals are used to generate the tree, the choice for the more power-efficient variant implies lower performance for one or more other metrics (in this case e.g. dynamic range).

It is also reassuring to designers that while there were thousands of *possible* topologies, just 15 were returned. This is in line with many analog designers' expectation that just a couple dozen opamp topologies serve most purposes. The challenge, of course, is which topologies those are, and for what specifications they are appropriate; which is what this tree makes clear.

It is important to remember that the tree is a classifier at its core, so one must avoid reading *too* much into it, such as the meaning of the exact values of the performance split values. In many cases the split value could increase or decrease by a few percent with no effect on classification. There are CART extensions to capture sensitivities to split values, but this is at a cost of additional complexity in the reported tree. Another extension is to let the user give preference to choosing certain split variables first, which may result in interesting alternative trees. We leave both to future work.

An additional benefit of tree extraction is based on there being more than 2-3 objectives, which means the raw data is difficult to visualize; the tree gives alternate perspective among 5 objectives, highlighting which topologies cover which performance regions.

4. Global Nonlinear Sensitivity Analysis

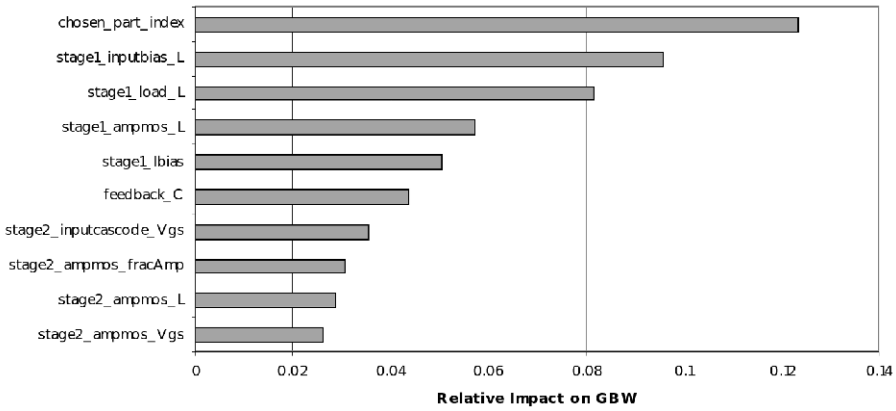


Figure 8-4. Global nonlinear sensitivity of GBW with respect to topology, sizing, and biasing variables, for 10 most important variables.

Introduction

The aim here is to address questions such as: "how much does each topology choice matter? Should I be changing the topology or device sizes? *Which* block or variables should I change?" There may even be more specific questions, such as, "How much does cascoding affect gain?"

Problem Formulation and Approach

Our approach to handle such questions is to perform global nonlinear sensitivity analysis. We need to be global – across the range of variables – because we have thousands of training points, and one cannot do small perturbations on integer-valued design variables such as topology-choice variables. We cannot assume linear because not being local means a Taylor approximation does not apply; topology-choice variables are categorical; and small ad-hoc tests showed that linear models fit poorly.

The sensitivity extraction flow we follow for each performance metric y is:

1. Given: a MOJITO-generated Pareto Optimal Set $Z = \{\phi_1^*, \phi_2^*, \ldots, \phi_{N_Z}^*\}$. Let $\boldsymbol{X} = \{\boldsymbol{x_j}\}$, $j = 1..N_Z$, where $\boldsymbol{x_j}$ is an N_d-dimensional design *vector* corresponding to design point ϕ_j. Let $\boldsymbol{y} = \{y_j\}$ where y_j is the corresponding scalar performance value of ϕ_j for the target objective (e.g. GBW)

2. Build a nonlinear regression model ψ that maps $\boldsymbol{X}$ to $\boldsymbol{y}$, i.e. $\hat{y} = \psi(\boldsymbol{x})$

Table 8-1. Procedure ModelSensitivities().

Inputs: $\boldsymbol{X}$, $\boldsymbol{y}$, N_d, ψ, N_{scr}
Outputs: $\boldsymbol{\zeta}$
1. for i = 1 to N_d:
2. $\quad \zeta_i = 0$
3. $\quad$ Repeat N_{scr} times:
4. $\quad\quad \boldsymbol{X_{scr}} = \boldsymbol{X}$ except randomly permute row i (for d_i)
5. $\quad\quad \boldsymbol{y_{scr}} = \psi(\boldsymbol{X_{scr}})$ # simulate model
6. $\quad\quad \zeta_i = \zeta_i + nmse(\boldsymbol{y}, \boldsymbol{y_{scr}})$
7. $\quad \zeta_{sum} = \sum_{i=1}^{d} \zeta_i$
8. $\quad \zeta_i = \frac{\zeta_i}{\zeta_{sum}}, i = 1..N_d$
9. $\quad$ Return $\boldsymbol{\zeta} = \zeta_i, i = 1..N_d$

3. From ψ, compute *global* sensitivities $\boldsymbol{\zeta} = \{\zeta_i\}, i = 1..N_d$

4. Return $\boldsymbol{\zeta}$

Steps 2 and 3 have specific challenges. Step 2, regressor construction, needs to handle numerical *and* categorical input variables. (Categorical variables are those which have discrete values and no relation among the discrete values, such as topology choice variables). This prevents usage of polynomials, splines / piecewise polynomials. support vector machines, kriging, and neural networks. GP-based symbolic regression such as CAFFEINE (McConaghy and Gielen, 2006) handles categorical variables, but it would run very slowly on 50 input variables and 1500 training samples. A CART tree is not appropriate because the model needs to do regression, not classification. However, a relatively recent technology achieves the effect of regression on CART trees by boosting them: stochastic gradient boosting (SGB) (Friedman, 2002). SGB also has acceptable scaling and prediction properties, so we employ it here.

Step 3 above needs to compute sensitivities from the model, yet be global, nonlinear, and ideally, nonparametric. The proposed solution defines global nonlinear sensitivity *impact* for a variable d_i as the relative error that a scrambled input variable d_i will give in predicting, compared to other variables $d_j, j = 1..d, j \neq i$ when they are scrambled. Table 8-1 gives the algorithm *ModelSensitivities()* that uses this concept to extract impacts (inspired by chapter 10 of (Hastie et al., 2001)). For each variable (line 1), it does repeated scrambling (lines 3-4) and keeps track of the resulting model error (lines 5-6). It normalizes the results (line 6-7) and returns. N_{scr} is number of scrambles; $nmse$ is normalized mean-squared error.

Results and Discussion

With this proposed flow, we extracted global nonlinear sensitivities for each performance. SGB and CART were coded in about 500 lines of Python programming language. SGB parameters were: learning rate α=0.10, minimum tree depth = 2, maximum tree depth = 7, target training error 5%. N_{scr} = 500. Build time for the SGB on modeling the objective GBW was about 15 s on a 2.0 GHz Linux machine, returning a model containing 282 CARTs. Impact extraction from the model took about 25 s.

Figure 8-4 illustrates results the ten variables that impact GBW most. We see that the most important variable is $chosen\_part\_index$, which selects one vs. two stages. The variables that are commonly associated with the GBW of opamps – the bias current of the first stage and the size of the compensation capacitance – also show up. Interestingly, the figure also indicates a large influence of the length of the transistors in the first stage (input, folding and load). This can be readily explained: these lengths directly influence the impedance on the internal nodes, and hence the location of the non-dominant pole. The phase margin requirement ($> 65°$) translates into the requirement that this non-dominant pole frequency is sufficiently higher than the GBW (approx 2x) (Sansen, 2006). It is also interesting to see that for GBW, only one topology parameter made it into the top ten variables; sizing parameters comprise the other nine. This means that once one vs. two stages is chosen, changing the right sizing variables will make the biggest difference to GBW. Of course, the most sensitive variables can be different for different performance metrics, and the designer must consider all metrics.

5. Extraction of Analytical Performance Tradeoffs

Introduction

Designers often they manually manipulate equations that relate performance tradeoffs (Razavi, 2000). Equations facilitate understanding because a direct relationship is expressed and the model is manipulatable to change the output variable. The problem is that hand-derived analytical expressions are based on 1^{st} or 2^{nd} order approximations may have little relation to the process technology, possibly having error of 20% or 50% or more. Some recent work has hinted towards automation. (Smedt and Gielen, 2003) did a single-topology multi-objective optimization run, then generated blackbox model of performance tradeoffs; but unfortunately the model is blackbox (giving no insight) and that only a single topology does not adequately describe the capabilities of the process technology. (Vogels and Gielen, 2003) conducted a thorough manual search of A/D converter technical publications to get Pareto Front data across many topologies, then created a whitebox model the performance trade-

offs. This, of course, was highly time-consuming, is already obsolete due to new process technology, and the models themselves were restricted to a fixed template.

The aims of this section are to (1) automatically extract analytical performance-tradeoff equations that are (2) in agreement with process technology, (3) span a huge set of possible topologies, (4) without being restricted to a predefined functional template, and (5) can be automatically generated with each new process.

Approach

To meet the aims, we propose the following approach:

1. Given: a MOJITO-generated Pareto Optimal Set, $Z = \{\phi_1^*, \phi_2^*, \ldots, \phi_j^*, \ldots, \phi_{N_Z}^*\}$. Let $\boldsymbol{Y} = \{\boldsymbol{y_j}\}$, $j = 1..N_Z$, where $\boldsymbol{y_j} = \boldsymbol{f_j}(\phi_j) = \{f_1(\phi_j^*), f_2(\phi_j^*), \ldots, f_{N_f}(\phi_j^*)\}$. $\boldsymbol{Y} \in \Re^{N_f x N_Z}$.

2. Choose a target performance objective (e.g. GBW) indexed by I; $I \in \{1, 2, \ldots, i, \ldots, N_f\}$.

3. Let $\boldsymbol{X_{-I}}$ be all rows in $\boldsymbol{Y}$ except I. Let $\boldsymbol{y}$ be the I^{th} row of $\boldsymbol{Y}$. Therefore $\boldsymbol{y}$ has the data of the target objective, and $\boldsymbol{X_{-I}}$ has the data of the remaining objectives.

4. Build a set of symbolic regression models, M, which map $\boldsymbol{X_{-I}}$ to $\boldsymbol{y}$, trading off model error vs. model complexity.

5. Return the resulting model(s) M for inspection by the user.

We can use whichever competent symbolic regression system we wish, with the main constraint that the resulting models must be interpretable, and with good prediction accuracy. We use CAFFEINE (McConaghy and Gielen, 2006), which has the distinguishing feature that it follows a grammar of canonical-form functions (sum of products, of sums). This grammar ensures that the resulting expressions are highly interpretable. CAFFEINE is multi-objective, generating a Pareto Optimal set of model complexity vs. modeling error. CAFFEINE settings were: N_B = maximum number of basis functions = 15, N_{pop} = population size = 200, $N_{gen,max}$ = 5000 generations, and maximum tree depth = 8. All operators had equal probability, except parameter mutation was 5x more likely. The runtime was about 10 minutes a 2.5 GHz Linux machine.

Results and Discussion

Table 8-2 shows results for GBW. We expected gain to be strongly related to the GBW, and it turns out that a simple linear relation between the two will

Table 8-2. Whitebox models Capturing Performance Tradeoff.

Train error	$log(GBW)$ **Expression**
8.7 %	$10.28 - 0.049 * gain$
7.3 %	$5.65 + 86.5/gain + (2.92e\text{-}11) * slewrate$
6.8 %	$5.72 + 80.2/gain + (4.75e\text{-}6) * \sqrt{slewrate}$
5.7 %	$7.30 + 47.76/gain - (3.430e\text{+}3)/\sqrt{slewrate}$
4.1 %	$4.48 + 24.9/\sqrt{gain} - (8.60e\text{+}6)/(gain^2 * \sqrt{slewrate})$

get $< 9\%$ training error. That is, a linear relation with gain will explain all but 9% of the variation of GBW. But for a better fit, i.e. to explain the variation with better resolution, more complex nonlinear relations are needed, leading to an inverse relationship of GBW with $\sqrt{(gain)}$. The slew rate objective is also needed for a reasonable model. Interestingly, dynamic range and power are not needed to get within 4.1% training error. Cross-examination with the scatterplots (Figure 8-2) confirms that the strongest tradeoffs are indeed among gain, GBW, and slew rate.

6. Conclusion

This chapter presented a methodology to help designers maintain their expert insights on the structure-parameters-fitness relationship; which for circuit design translates to topology-sizings-performances. The approach is to take a data mining perspective on a Pareto Optimal Set of structures: extract a specifications-to-structure decision tree (via CART); do global nonlinear sensitivity analysis on structure and parameter variables (via SGB and a variable-scrambling heuristic); and generate analytical whitebox models to capture tradeoffs among objective function values (via GP symbolic regression - CAFFEINE). These approaches are all complementary as they answer different designer questions. Once extracted, the knowledge can readily be distributed to other designers, without need for more synthesis. Results are shown for operational amplifier circiut design on a database containing thousands of Pareto Optimal designs across five objectives. As a final note, we must emphasize once again that these techniques are meant to augment designer experience, not replace it. The designer is key.

7. Acknowledgment

Funding for the reported research results is acknowledged from IWT/Medea+ Uppermost, Solido Design Automation Inc. and FWO Flanders.

References

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984). *Classification and Regression Trees*. Chapman & Hall.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. Evolutionary Computation*, 6(2):182–197.

Friedman, J.H. (2002). Stochastic gradient boosting. *Journal of Computational Statistics & Data Analysis*, 38(4):367–378.

Harjani, Ramesh, Rutenbar, Rob A., and Carley, L. Richard (1989). Oasys: A framework for analog circuit synthesis. (12):1247–1266.

Hastie, T., Tibshirani, R., and Friedman, J.H. (2001). *The Elements of Statistical Learning*. Springer.

Hornby, Gregory S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 815–822, Seattle, Washington, USA. ACM Press.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

McConaghy, Trent and Gielen, Georges (2005). Genetic programming in industrial analog CAD: Applications and challenges. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 19, pages 291–306. Springer, Ann Arbor.

McConaghy, Trent and Gielen, Georges (2006). Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 855–862, Seattle, Washington, USA. ACM Press.

McConaghy, Trent, Palmers, Pieter, Gielen, Georges, and Steyaert, Michiel (2007). Genetic programming with reuse of known designs. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 10, pages 161–186. Springer, Ann Arbor.

Razavi, B. (2000). *Design of Analog CMOS Integrated Circuits*. McGraw-Hill.

Sansen, W. (2006). *Analog Design Essentials*. Springer.

Smedt, Bart De and Gielen, Georges G.E. (2003). Watson: Design space boundary exploration and model generation for analog and rfic design. 22(2):213–224.

Vogels, Martin and Gielen, Georges G.E. (2003). Architectural selection of a/d converters. In *Proceedings of Design Automation Conference*, pages 974–977.

Williams, J., editor (1991). *Analog Design: Art, Science, and Personalities*. Newnes Press.

Chapter 9

DOES COMPLEXITY MATTER? ARTIFICIAL EVOLUTION, COMPUTATIONAL EVOLUTION AND THE GENETIC ANALYSIS OF EPISTASIS IN COMMON HUMAN DISEASES

Jason H. Moore[1], Casey S. Greene[1], Peter C. Andrews[1] and Bill C. White[1]
[1]*Dartmouth College, One Medical Center Drive HB7937, Lebanon, NH 03756 USA.*

Abstract Common human diseases are complex and likely the result of nonlinear interactions between multiple different DNA sequence variations. One goal of human genetics is to use data mining and machine learning methods to identify sets of discrete genetic attributes that are predictive of discrete measures of health in human population data. A variety of different computational intelligence methods based on artificial evolution have been developed and applied in this domain. While artificial evolution approaches such as genetic programming show promise, they are only loosely based on real biological and evolutionary processes. It has recently been suggested that a new paradigm is needed where "artificial evolution" is transformed to "computational evolution" by incorporating more biological and evolutionary complexity into existing algorithms. Computational evolution systems have been proposed as more likely to solve problems of interest to biologists and biomedical researchers. To test this hypothesis, we developed a prototype computational evolution system for the analysis of human genetics data capable of evolving operators of arbitrary complexity. Preliminary results suggest that more complex operators result in better solutions. Here we introduce modifications including a simpler stack-based solution representation, the ability to maintain and use an archive of solution building blocks, and a simpler set of solution operator building blocks capable of learning to use pre-processed expert knowledge. A parameter sweep suggests that operators that can use expert knowledge or archival information outperform those that cannot. This study supports the idea that complexity matters and thus the consieration of computational evolution for bioinformatics problem-solving in the domain of human genetics.

Keywords: genetic epidemiology, symbolic discriminant analysis

1. Introduction

Computational Challenges in Human Genetics

The domain of human genetics is rapidly changing due to the availability of new technologies that facilitate the measurement of more than 10^6 DNA sequence variations across the genome. As such, we are no longer limited by our ability to measure the genome. Rather, we are now significantly limited by the lack of computational and statistical tools for identifying genetic measures that play a role in the initiation, progression and severity of common human diseases such as breast cancer and schizophrenia. For the purposes of this paper we will focus exclusively on the single nucleotide polymorphism or SNP which is a single nucleotide or point in the DNA sequence that differs among people. Most SNPs have two alleles (e.g. A or a) that combine in the diploid human genome in one of three possible genotypes (e.g. AA, Aa, aa). It is anticipated that at least one SNP occurs approximately every 100 nucleotides across the 3×10^9 nucleotide human genome making it the most common type of genetic variation. An important goal in human genetics is to determine which of the many hundreds of thousands of informative SNPs are useful for predicting who is at risk for common diseases.

The charge for computer science and bioinformatics is to develop algorithms for the detection and characterization of those SNPs that are predictive of human health and disease. Success in this genome-wide endeavor will be difficult due to nonlinearity in the genotype-to-phenotype mapping relationship that is due, in part, to epistasis or nonadditive gene-gene interactions. Epistasis was recognized more than 100 years ago as playing an important role in the mapping between genotype and phenotype (Bateson, 1909). Today, this idea prevails and epistasis is believed to be a ubiquitous component of the genetic architecture of common human diseases (Moore, 2003). As a result, the identification of genes with genotypes that confer an increased susceptibility to a common disease will require a research strategy that embraces, rather than ignores, this complexity (Moore, 2003; Moore and Williams, 2005; Thornton-Wells et al., 2004). The implication of epistasis from a data mining point of view is that SNPs need to be considered jointly in learning algorithms rather than individually. Because the mapping between the attributes and class is nonlinear, the concept difficulty is high. The challenge of modeling attribute interactions has been previously described (Freitas, 2001). The goal of the present study is to develop a machine learning strategy for detecting and characterizing gene-gene interactions using the latest methods and ideas from evolutionary computing.

Table 9-1. Penetrance values for genotypes from two SNPs.

	AA (0.25)	Aa (0.50)	aa (0.25)
BB (0.25)	0	1	0
Bb (0.50)	1	0	1
bb (0.25)	0	1	0

A Simple Example of the Concept Difficulty

Epistasis or gene-gene interaction can be defined as biological or statistical (Moore and Williams, 2005). Biological epistasis occurs at the cellular level when two or more biomolecules physically interact. In contrast, statistical epistasis occurs at the population level and is characterized by deviation from additivity in a linear mathematical model. Consider the following simple example of statistical epistasis in the form of a penetrance function. Penetrance is simply the probability (P) of disease (D) given a particular combination of genotypes (G) that was inherited (i.e. $P[D|G]$). A single genotype is determined by one allele (i.e. a specific DNA sequence state) inherited from the mother and one allele inherited from the father. For most single nucleotide polymorphisms or SNPs, only two alleles (encoded by A or a) exist in the biological population. Therefore, because the order of the alleles is unimportant, a genotype can have one of three values: AA, Aa or aa. The model illustrated in Table 9-1 is an extreme example of epistasis. Let's assume that genotypes AA, aa, BB, and bb have population frequencies of 0.25 while genotypes Aa and Bb have frequencies of 0.5 (values in parentheses in Table 9-1). What makes this model interesting is that disease risk is dependent on the particular combination of genotypes inherited. Individuals have a very high risk of disease if they inherit Aa or Bb but not both (i.e. the exclusive OR function). The penetrance for each individual genotype in this model is 0.5 and is computed by summing the products of the genotype frequencies and penetrance values. Thus, in this model there is no difference in disease risk for each single genotype as specified by the single-genotype penetrance values. This model was first described by Li and Reich (Li and Reich, 2000). Heritability, or the size of the genetic effect, is a function of these penetrance values. In this model, the heritability is 1.0, the maximum possible, because the probability of disease is completely determined by the genotypes at these two DNA sequence variations. All the heritability in this model is due to epistasis. As Freitas reviews, this general class of problems has high concept difficulty (Freitas, 2002).

Artificial and Computational Evolution

Numerous machine learning and data mining methods have been developed and applied to the detection of gene-gene interactions in population-based studies of human disease. These include, for example, traditional methods such as neural networks (Lucek and Ott, 1997) and novel methods such as multifactor dimensionality reduction (Ritchie et al., 2001). Evolutionary computing methods such as genetic programming (GP) have been applied to both attribute selection and model discovery in the domain of human genetics. For example, Ritchie et al (Ritchie et al., 2003) used GP to optimize both the weights and the architecture of a neural network for modeling the relationship between genotype and phenotype in the presence of gene-gene interactions. More recently, GP has been successfully used for both attribute selection (Moore and White, 2006; Moore and White, 2007a; Moore, 2007; Greene et al., 2007) and genetic model discovery (Moore et al., 2007).

Genetic programming is an automated computational discovery tool that is inspired by Darwinian evolution and natural selection (Banzhaf et al., 1998a; Koza, 1992; Koza, 1994; Koza et al., 1999; Koza et al., 2003; Langdon, 1998; Langdon and Poli, 2002). The goal of GP is to evolve computer programs to solve problems. This is accomplished by first generating random computer programs composed of the building blocks needed to solve or approximate a solution. Each randomly generated program is evaluated and the good programs are selected and recombined to form new computer programs. This process of selection based on fitness and recombination to generate variability is repeated until a best program or set of programs is identified.

Genetic programming and its many variations have been applied successfully to a wide range of different problems including data mining and knowledge discovery (e.g. (Freitas, 2002)) and bioinformatics (e.g. (Fogel and Corne, 2003)). Despite the many successes, there are a large number of challenges that GP practitioners and theorists must address before this general computational discovery tool becomes one of several tools that a modern problem solver calls upon (Yu et al., 2006). Spector, as part of an essay regarding the roles of theory and practice in genetic programming, discusses the push towards biology by GP practicioners (Spector, 2003). Banzhaf et al. propose that overly simplistic and abstracted artificial evolution (AE) methods such as GP need to be transformed into computational evolution (CE) systems that more closely resemble the complexity of real biological and evolutionary systems (Banzhaf et al., 2006). Evolution by natural selection solves problems by building complexity. As such, computational systems inspired by evolution should do the same. The working hypothesis addressed in the present study is that a GP-based genetic analysis system will find better solutions faster if it is implemented as a CE system that can evolve a variety of complex operators that in turn generate

variability in solutions. This is in contrast to an AE system that uses a fixed set of operators.

Research Questions Addressed and Overview

We have previously developed a prototype CE system and have shown that it is capable of evolving complex operators for problem solving in human genetics (Moore et al., 2008). The goal of the present study was to extend and evaluate this new open-ended computational evolution system for the detection and characterization of epistasis or gene-gene interactions that are associated with risk of human disease. New features include simpler operator building blocks, list-based solutions with stack-based evaluation and an attribute archive that provides the system with a feedback loop between the population of solutions and the solution operators. These new features are consistent with the idea of transforming an AE system to a CE system.

2. A Computational Evolution System

Our primary goal was to develop, extend and evaluate a computational evolution system that is capable of open-ended evolution for bioinformatics problem-solving in the domain of human genetics. Figure 9-1 gives a graphical overview of our hierarchically-organized and spatially-extended GP system that is capable of open-ended computational evolution. At the bottom layer of this hierarchy is a grid of solutions. At the second layer of the hierarchy is a grid of operators of any size and complexity that are capable of modifying the solutions (i.e. solution operators). At the third layer in the hierarchy is a grid of mutation operators that are capable of modifying the solution operators. At the highest level of the hierarchy is the mutation frequency that determines the rate at which operators are mutated. An attribute archive provides a feedback loop between the solutions and the solution operators. Expert knowledge in the form of pre-processed ReliefF scores for the attributes is also provided. The details of the experimental design used to evaluate this system are described in Section 3.

Problem Solutions: Their Representation, Fitness Evaluation and Reproduction

The goal of a classifier is to accept as input two or more discrete attributes (i.e. SNPs) and produce a discrete output that can be used to assign class (i.e. healthy or sick). Here, we used symbolic discriminant analysis or SDA as our classifier. The SDA method (Moore et al., 2002) has been described previously for this problem domain (Moore et al., 2008; Moore et al., 2007; Moore and White, 2007a). SDA models consist of a set of attributes and constants as input and a set of mathematical functions that produce for each instance in the

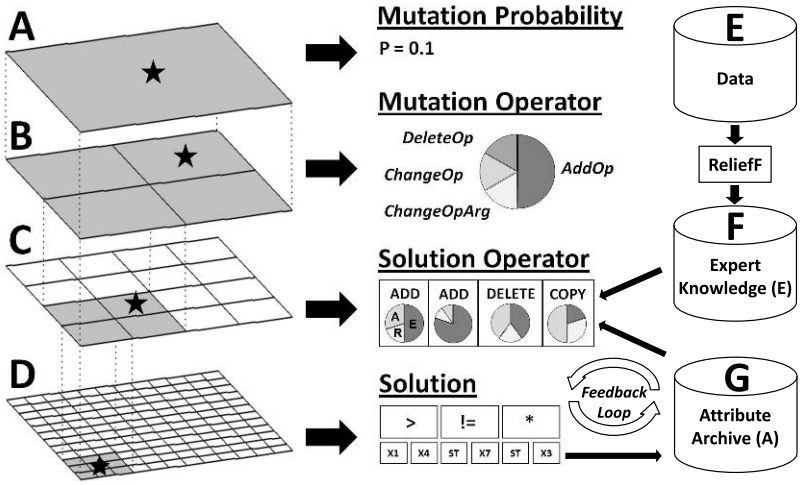


Figure 9-1. Visual overview of our computational evolution system for discovering symbolic discriminant functions that differentiate disease subjects from healthy subjects using information about single nucleotide polymorphisms (SNPs). The hierarchical structure is shown on the left while some specific examples at each level are shown in the middle. At the lowest level (D) is a grid of solutions. Each solution consists of a list of functions and their arguments (e.g. X1 is an attribute) that are evaluated using a stack (denoted by ST in the solution). The next level up (C) is a grid of solution operators that each consists of some combination of the ADD, DELETE and COPY functions each with their respective set of probabilities that define whether expert knowledge from ReliefF (denoted by E in the probability pie) or the archive (denoted by A in the probability pie) are used instead of a random generator (denoted by R in the probability pie). ReliefF scores are derived by pre-processing the data (E) and are stored for use by the system (F). The attribute archive (G) is derived from the frequency with which each attribute occurs among solutions in the population. The top two levels of the hierarchy (A and B) exist to generate variability in the operators that modify the solutions. This system allows operators of arbitrary complexity to modify solutions. Note that we used 18x18 and 36x36 grids of 324 and 1296 solutions, respectively, in the present study. A 12x12 grid is shown here as an example.

data set a score called a symbolic discriminant score. Here, our SDA function set was $+, -, *, /, \%, <, <=, >, >=, ==, != $ where the $\%$ operator is a mod operation and $/$ is a protected division. The SDA models are represented as a list of expressions here instead of as expression trees as has been used in the past to facilitate stack-based evaluation of the classifiers and to facilitate their representation in text files. This is similar to the GP implementation using

arrays and stack as described by Keith and Martin (Keith and Martin, 1994), Perkis (Perkis, 1994), and Banzaf et al. (Banzhaf et al., 1998b).

Classification of instances into one of the two classes requires a decision rule that is based on the symbolic discriminant score. Thus, for any given symbolic discriminant score (S_{ij}) in the ith class and for the j^{th} instance, a decision rule can be formed such that if $S_{ij} > S_o$ then assign the instance to one class and if $S_{ij} <= S_o$ then assign the observation to the other class. When the prior probability that an instance belongs to one class is equal to the probability that it belongs to the other class, S_o can be defined as the arithmetic mean of the median symbolic discriminant scores from each of the two classes. This is the classification rule we used in the present study and is consistent with previous work in this domain (Moore et al., 2008; Moore et al., 2007; Moore and White, 2007a). Using this decision rule, the classification accuracy for a particular discriminant function can be estimated from the observed data. Here, accuracy is defined as (TP + TN)/(TP + TN + FP + FN) where TP are true positives (TP), TN are true negatives, FP are false positives, and FN are false negatives. We used accuracy as the fitness measure for SDA solutions as has been described previously but lightly weight it such that for solutions with equivalent accuracy, ones with shorter genome sizes are preferable (Moore et al., 2008; Moore et al., 2007; Moore and White, 2007a).

All SDA solutions in a population are organized on a toroidal grid with specific X and Y coordinates (see example in Figure 9-1). As such, they resemble previous work on cellular genetic programming (Folino et al., 1999). In the present study we evaluated grid sizes of 18x18 and 36x36 for total population sizes of 324 and 1296, respectively. Reproduction of solutions in the population is handled in a spatial manner. Each solution is considered for reproduction in the context of its Moore neighborhood using an elitist strategy. That is, each solution in question will compete with its eight neighbors and be replaced in the next generation by the neighbor with the highest fitness. This combines ideas of tournament selection that is common in GP with a set of solutions on a grid. Variability in solutions is generated using hierarchically organized operators. This is described below.

Operators for Computational Evolution: Generating Solution Variability

Traditional artificial evolution approaches such as GP use a fixed set of operators that include mutation and recombination, for example. The goal of developing a computational evolution system was to provide building blocks (i.e. simple functions) for operators that could be combined to create new operators. We started with the following three basic operator building blocks. The first operator building block, ADD, adds a new function and its arguments

to the list of functions and arguments that comprise a solution. The second operator building block, DELETE, deletes a function from the list of functions. The third operator, COPY, copies a function from the list of functions from the solution in focus to a solution located within the Moore neighborhood. These operators can combine in any number and order to generate solution operators of arbitrary complexity. The mutation operators described below increase or decrease the size and content of the solution operators.

Each of the operator building blocks has a vector of three probabilities associated with it. The first number specifies the probability that the function that is added, deleted or copied to a solution is determined stochastically. The second specifies the probability that the function that is added, deleted or copied to a solution is determined according to an archive of attributes that is ranked according to the frequency that they occur in the population of solutions (see below). The third specifies the probability that the function that is added, deleted or copied to a solution is determined according to ReliefF scores for the attributes (see below). The ability to use expert knowledge (i.e. environmental sensing) is important in this domain. For example, pre-processed ReliefF scores have been shown to improve the performance of GP as a wrapper in this domain when used in a multiobjective fitness function (Moore and White, 2007a), when used to guide recombination (Moore and White, 2006) and when used to guide mutation (Greene et al., 2007). This is consistent with Goldberg's ideas about exploiting good building blocks in competent genetic algorithms (Goldberg, 2002) and provides a source of complexity as recommended by Banzhaf et al. (Banzhaf et al., 2006). For example, the use of the archive creates a feedback loop between the solutions and the solution operators.

As with the solutions, each operator is organized on a toroidal grid with a specific X and Y coordinate. We assigned each operator to a set of solutions. This allows for averaging an operator's positive or negative effects on multiple solutions. In this study, we assigned each operator to a 3x3 grid of nine solutions. Thus, the population of solution operators is organized in a 6x6 grid when an 18x18 grid is used for the solutions and 12x12 when a 36x36 grid is used for the solutions. The assignment of fitness to solution operators is based on Edmonds's Meta-GP framework (Edmonds, 1998; Edmonds, 2001). The fitness for an operator is the average proportionate change in fitness of the classifiers it operated on smoothed over preceeding generations and with a strong fitness penalty for operators that do not change the fitness of solutions. The smoothing is accomplished by decaying by a factor of two, the previous generation's fitness, and adding that to the fitness for this generation.

Mutation of Operators for Computational Evolution: Generating Operator Variability

An important goal for the computational evolution system is the ability to generate variability in the operators that modify solutions. To accomplish this goal we developed an additional level in the hierarchy (Figure 9-1) with mutation operators that specifically alter the operators described above. We defined four different fixed mutation operators that are each assigned to a 2x2 grid of solution operators. Solution operators can be modified in the following four ways. First, an operator can have a specific operator building block deleted (DeleteOperator). Second, an operator can have a specific operator building block added (AddOperator). Third, an operator can have a specific operator building block changed (ChangeOperator). Finally, an operator can have its arguments changed (ChangeOperatorArguments). This latter function allows, for example, the range that a DeleteRangeOperation would use. In this study, we initialized the probabilities with which each the these mutation operators are used to 0.25. These are randomly regenerated at a frequency equal to the overall mutation probability (see below) and their fitness is determined by the change in fitness of the solution operators that they act on.

Mutation Frequency

The top level of the computational evolution system hierarchy (see Figure 9-1) is the mutation frequency that controls the probability that one of the four mutation sets in the next level down will mutate a given solution operator two levels down. In the present study we fixed this to 0.1 or 0.5. In the future this will be an evolvable parameter. This frequency does not control the frequency with which a solution operator modifies a solution. That is controlled by the operator when it specifies which solution(s) it will modify.

Environmental Sensing Using an Archive

Previous studies have demonstrated the utility of archiving GP results for reuse (Vladislavleva et al., 2007). We have implemented here an archive that ranks the attributes by the frequency with which they appear in solutions from the population. These are ranked by their frequency and then used by the ADD, DELETE and COPY operators to decide what gets added, deleted or copied. We used here a cumulative archive that updates the previous results each generation. The archive is an important part of the complexity of the CE system because it provides a feedback loop between the solutions and the solution operators.

Environmental Sensing Using ReliefF

As mentioned above, the use of expert knowledge is important for the application of GP strategies to solving complex problems in human genetics. Here, we used pre-processed ReliefF scores for all of the attributes in the dataset as a source of statistical knowledge for the analysis. Kira and Rendell developed the Relief algorithm that is capable of detecting attribute dependencies (Kira and Rendell, 1992). Relief estimates the quality of attributes through a type of nearest neighbor algorithm that selects neighbors (instances) from the same class and from the different class based on the vector of values across attributes. Weights (W) or quality estimates for each attribute (A) are estimated based on whether the nearest neighbor (nearest hit, H) of a randomly selected instance (R) from the same class and the nearest neighbor from the other class (nearest miss, M) have the same or different values. This process of adjusting weights is repeated for m instances. The algorithm produces weights for each attribute ranging from -1 (worst) to +1 (best). Kononenko improved upon Relief by choosing n nearest neighbors instead of just one (Kononenko, 1994). This new ReliefF algorithm has been shown to be more robust to noisy attributes and is widely used in data mining applications. We have developed a modified ReliefF algorithm for the domain of human genetics called Tuned ReliefF (TuRF). We have previously shown that TuRF is significantly better than ReliefF in this domain (Moore and White, 2007b). The TuRF algorithm systematically removes attributes that have low quality estimates so that the ReliefF values if the remaining attributes can be re-estimated. We applied TuRF as described by Moore and White (Moore and White, 2007b) to the data set analyzed and provided the results to the CE system as expert knowledge that can then used by the ADD, DELETE and COPY operators to decide what gets added, deleted or copied.

Implementation

The computational evolution system described above was programmed entirely in C++. A single run of the system with a population of 1296 solutions on a 36x36 grid for 1000 generations took approximately 10 minutes on a 3.0 GHz AMD Opteron processor. Multiple runs for the experiments described below were carried out in parallel using 100 processors.

3. Experimental Design and Data Analysis

Our goal was to provide an evaluation of the CE system described above using a full factorial experimental design. The central question addressed in this study is whether the ability to evolve operators of arbitrary complexity improves the quality of the SDA models. More specifically, we were interested in whether

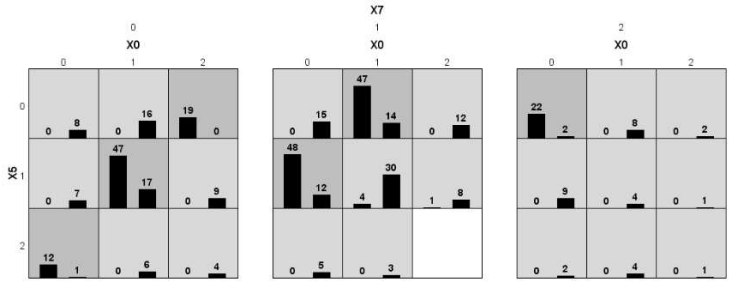


Figure 9-2. Distribution of healthy controls (left bars) and diseased cases (right bars) for each combination of genotypes (coded 0, 1 and 2) for the three functional attributes or SNPs (X0, X5 and X7). Note the nonlinear pattern of high-risk (dark grey) and low-risk (light grey) genotype combinations that is indicative of a nonlinear interaction.

use of the archive or the expert knowledge improves accuracy of the solutions over that provided by random selection of attributes when new functions were added, deleted or copied by the solution operators. We considered four different factors in this study. First, to evaluate the question posed above, we evaluated the effect of using the archive or expert knowledge from ReliefF on accuracy and compared this with purely stochastic decisions. Second, we evaluated the effect of the grid size (18x18 and 36x36) and the number of generations (100, 500 and 1000) on the quality of the solutions. Third, we evaluated the effect of the mutation frequency (0.1 and 0.5) on the accuracy of the solutions generated by the CE system. This generated a total of 36 factor combinations. A total of 100 runs with different random seeds were performed for each factor combination for a total of 3600 runs. For each run the best solution was selected and the accuracy of the classifier recorded. Analysis of variance (ANOVA) with Tukey's post-hoc analysis with correction for multiple testing were used to compare mean accuracies between the different factor levels and level combinations. All results were considered significant at the 0.05 level.

We used a simulated data set consisting of 20 total attributes (SNPs) and 400 instances (200 cases and 200 controls). Three of the 20 SNPs are associated with disease class through a nonlinear interaction as described in the introduction. This is a well-known data set that is distributed with the multifactor dimensionality reduction (MDR) software that was designed specifically for detecting gene-gene interactions (Ritchie et al., 2001). Figure 9-2 illustrates the distribution of healthy controls (left bars) and diseased cases (right bars) for each combination of genotypes (coded 0, 1 and 2) for the three functional attributes or SNPs (X0, X5 and X7). Note the nonlinear pattern of high-risk (dark grey) and low-risk (light grey) genotype combinations. The optimal classification of this dataset yields a classification accuracy of approximately 0.87.

4. Results

Figure 9-3 summarizes the distribution of classifier accuracies obtained from running the computational evolution system 100 times on the simulated data with each of the different parameter settings. The primary result of this analysis is that the use of both the archive and the pre-processed expert knowledge significantly improved performance of the system as measured by accuracy of the best symbolic discriminant functions discovered in each run ($P < 0.0001$). Further, Tukey's post-hoc analysis showed that including the archive was better than random attribute selection ($P = 0.0001$) and that including expert knowledge was better than both random selection ($P < 0.0001$) and the archive

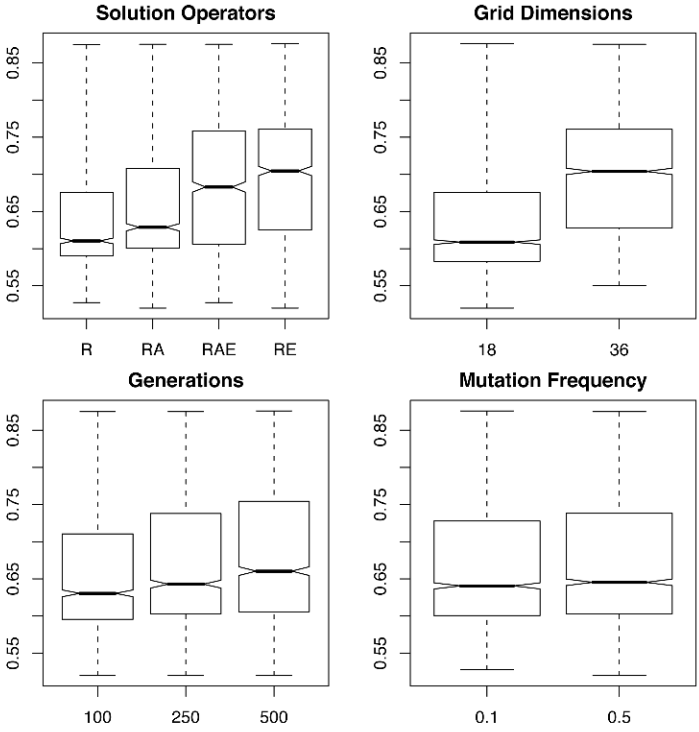


Figure 9-3. Distribution of classifier accuracies obtained from running the computational evolution system 100 times on the simulated data with each of the different parameter settings. Solution operators levels included random (R), random plus archive (RA), random plus archive plus expert knowledge (RAE), and random plus expert knowledge (E). Grid dimension levels included 18x18 or 324 solutions (18) and 36x36 or 1296 solutions (36). Generation levels included 100, 250 and 500. Mutation frequency levels included 0.1 and 0.5.

($P < 0.0001$). These results suggest that the CE system was able to evolve solution operators that could exploit information in the archive and in the source of expert knowledge thus demonstrating complexity improves performance.

In addition, we found, as expected, that the size of the solution grid (i.e. population size) and the number of generations had a significant effect on the accuracy of the solutions discovered ($P < 0.0001$). Increased population size and increased number of generations yielded better solutions. We also evaluated whether increased mutation frequency for the operators improved performance. We found that a mutation frequency of 0.5 was not significantly better than 0.1 ($P > 0.1$). A statistically significant interaction was observed between grid size and the presence of expert knowledge for solution operators ($P < 0.001$). Tukey's analysis suggests that there is a nonadditive effect of using expert knowledge with increased grid size.

5. Discussion and Conclusions

Traditional artificial evolution systems such as GP may not be suitable for solving many complex problems. This is especially true in the domain of human genetics where expert knowledge about the problem is critical (Moore, 2007). Banzhaf et al. have suggested that artificial evolution methods such as GP will greatly benefit from our current understanding of the complexity of biological and evolutionary systems (Banzhaf et al., 2006). They propose a new research agenda in which computational evolution systems that mimic the complexity of biological systems will replace the overly simplified artificial evolution systems that have been inspired by biology, but largely ignore the complexity of biological processes. The goal of the present study was to specifically address whether a computational evolution system capable of evolving more complex operators will find better solutions than an artificial evolution system in the domain of human genetics. To accomplish this goal we have previously developed a prototype computational evolution system that is both spatially and hierarchically organized and is capable of evolving operators of arbitrary size and complexity from a set of basic operator building blocks (Moore et al., 2008). The preliminary experimental results from this previous study demonstrated that the ability to evolve more complex operators did indeed improve the ability of the system to identify good genetic models. The goal of the present study was to extend and evaluate this system in a way that is consistent with the research agenda proposed by Banzhaf et al. (Banzhaf et al., 2006). New features include simpler operator building blocks, list-based solutions with stack-based evaluation and an attribute archive that provides the system with a feedback loop between the population of solutions and the solution operators.

An important aspect of this computational evolution system is the ability to evolve operators of any size and complexity that modify solutions in a spatially-

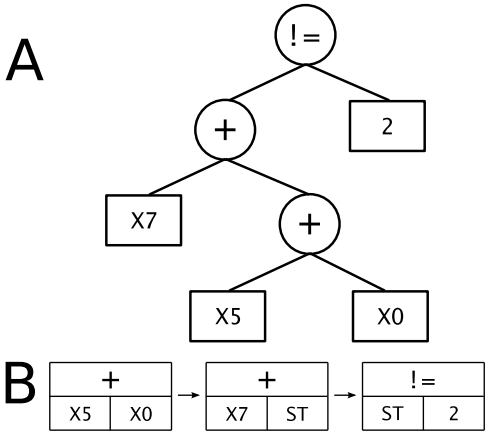


Figure 9-4. A highly predictive solution (accuracy = 0.8725) evolved by the CES shown in its tree representation (A) and stack representation (B) after silent portions have been pruned.

organized population. This is similar to real biological systems that evolve more complex genomic processes. For example, microRNAs that participate in post-translational regulation have evolved, in part, to help determine developmental processes such as body plan specification. Sempere et al. showed that the number of microRNAs an animal group has correlates strongly with the hierarchy of metazoan relationships (Sempere et al., 2007). The ability of species to evolve new biological processes plays an important role in increasing their complexity. Here, we provided the system three basic operator building blocks (ADD, DELETE and COPY) that could be pieced together in any number and combination. Further, we provided to each operator building block an argument consisting of several probabilities that specify how attributes should be selected for action by their specific operator. For example, when ADD is called to add a function to the list that comprises a solution, it must decide how to select this attribute. It can select the attribute randomly, probabilistically using the archive, or probabilistically using a source of expert knowledge. The probability of each of these scenarios can change over time by mutation and evolve according to whether this particular operator yields a positive change in fitness from one generation to the next in the solutions it modifies. Our results suggest that solution operators that evolve the ability to use the archive or the expert knowledge yield better solutions. We are not this first to develop a system that evolve its own operators. Spector developed an autoconstructive evolution system called PushGP that evolves its own reproduction and diversification machinery (Spector, 2001; Spector and Robinson, 2002). This study and others have laid the groundwork for the study presented here. The novel aspect of our system is the ability of the operators to learn to use other sources

of information, which is consistent with the idea of environmental sensing in biological organisms. This opens the door to using systems like this for solving complex problems in the biomedical sciences.

Our results suggest that the solution operators are able to evolve the ability to utilize information from a continuously updated attribute archive and pre-processed expert knowledge and that this ability leads to improved performance of the system. The ability to archive information about the frequency with which different attributes are represented in solutions and then to use that information to in turn guide which attributes are included in new solutions creates an important feedback loop in the system. Banzhaf et al. have noted the importance of feedback loops in complex biological systems (Banzhaf et al., 2006). This study supports that idea. An important question is whether there are other ways that this information or any other archive of information can be used to provide feedback loops similar to the one shown here or to other parts of the system. For example, a future goal is to generate an archive of functions to complement the archive of attributes for guiding the construction of solutions. Establishing and updating probability distributions for attributes and functions is similar to ideas from estimation of distribution algorithms (EDAs) where the solutions themselves are modeled and the models are used to generate new solutions. What other parts of the system could be targeted for feedback loops? One possibility would be to use the attribute archive to provide feedback to the mutation frequency. Convergence of the probabilities in the archive might indicate that the mutation frequency should be increased.

The ability of the system to use expert knowledge plays an important role in providing building blocks for solutions in a problem domain where there often aren't obvious building blocks. Here, we provided the system knowledge in form of pre-processed statistical information about attribute quality, however, the availability of biological knowledge about gene function, chromosomal location, biochemical pathways, etc. presents an opportunity to adapt this type of information to these learning algorithms. Future studies using a computational evolution system for the analysis of real data will have the opportunity to combine both statistical and biological knowledge.

Our future goal is to improve the computational evolution system by adding additional features that are inspired by the complexity of real biological systems. We will explore the use of archives with additional feedback loops to different part of the system. We will also make the overall mutation frequency at the highest level an evolvable parameter. The evolvability of the entire system will make it attractive to implement this system in parallel as an island model thus providing a virtual ecosystem with feedback between populations. Whether these additional features continue to improve the ability of this machine learning method to solve complex problems in human genetics still needs to be addressed.

Acknowledgment

This work was supported by National Institutes of Health (USA) grants LM009012, AI59694 and RR018787. We thank Dr. Wolfgang Banzhaf and the attendees of the 2007 Genetic Programming Theory and Practice (GPTP) Workshop for their insightful ideas about computational evolution.

References

Banzhaf, W., Beslon, G., Christensen, S., Foster, J. A., Kepes, F., Lefort, V., Miller, J., Radman, M., and Ramsden, J. J. (2006). From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics*, 7:729–735.

Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. (1998a). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA.

Banzhaf, Wolfgang, Poli, Riccardo, Schoenauer, Marc, and Fogarty, Terence C., editors (1998b). *Genetic Programming*, volume 1391 of *LNCS*, Paris. Springer-Verlag.

Bateson, W. (1909). *Mendel's Principles of Heredity*. Cambridge University Press, Cambridge.

Edmonds, Bruce (1998). Meta-genetic programming: Co-evolving the operators of variation. CPM Report 98-32, Centre for Policy Modelling, Manchester Metropolitan University, UK, Aytoun St., Manchester, M1 3GH. UK.

Edmonds, Bruce (2001). Meta-genetic programming: Co-evolving the operators of variation. *Elektrik*, 9(1):13–29. Turkish Journal Electrical Engineering and Computer Sciences.

Fogel, G.B. and Corne, D.W. (2003). *Evolutionary Computation in Bioinformatics*. Morgan Kaufmann Publishers.

Folino, Gianluigi, Pizzuti, Clara, and Spezzano, Giandomenico (1999). A cellular genetic programming approach to classification. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1015–1020, Orlando, Florida, USA. Morgan Kaufmann.

Freitas, A. (2001). Understanding the crucial role of attribute interactions. *Artificial Intelligence Review*, 16:177–199.

Freitas, A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer.

Goldberg, D. E. (2002). *The Design of Innovation*. Kluwer.

Greene, C. S., White, B. C., and Moore, J. H. (2007). An expert knowledge-guided mutation operator for genome-wide genetic analysis using genetic programming. *Lecture Notes in Bioinformatics*, 4774:30–40.

Keith, M. J. and Martin, M. C. (1994). *Advances in Genetic Programming*. MIT Press.

Kira, K. and Rendell, L. A. (1992). A practical approach to feature selection. *In: Machine Learning: Proceedings of the AAAI'92*.

Kononenko, I. (1994). Estimating attributes: Analysis and extension of relief. *Machine Learning: ECML-94*, pages 171–182.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, John R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Langdon, W. B. and Poli, Riccardo (2002). *Foundations of Genetic Programming*. Springer-Verlag.

Langdon, William B. (1998). *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston.

Li, W. and Reich, J. (2000). A complete enumeration and classification of two-locus disease models. *Human Heredity*, 50:334–49.

Lucek, P.R. and Ott, J. (1997). Neural network analysis of complex traits. *Genetic Epidemiology*, 14(6):1101–1106.

Moore, J. H. (2003). The ubiquitous nature of epistasis in determining susceptibility to common human diseases. *Human Heredity*, 56:73–82.

Moore, J. H. (2007). Genome-wide analysis of epistasis using multifactor dimensionality reduction: feature selection and construction in the domain of human genetics. In *Knowledge Discovery and Data Mining: Challenges and Realities with Real World Data*. IGI.

Moore, J. H. and White, B. C. (2006). Exploiting expert knowledge in genetic programming for genome-wide genetic analysis. *Lecture Notes in Computer Science*, 4193:969–977.

Moore, J. H. and White, B. C. (2007a). Genome-wide genetic analysis using genetic programming: The critical need for expert knowledge. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, Genetic and Evolutionary Computation. Springer.

Moore, J. H. and White, B. C. (2007b). Tuning relieff for genome-wide genetic analysis. *Lecture Notes in Computer Science*, 4447:166–175.

Moore, J. H. and Williams, S. W. (2005). Traversing the conceptual divide between biological and statistical epistasis: Systems biology and a more modern synthesis. *BioEssays*, 27:637–46.

Moore, J.H., Andrews, P.C., Barney, N., and White, B.C. (2008). Development and evaluation of an open-ended computational evolution system for the genetic analysis of susceptibility to common human diseases. *Lecture Notes in Computer Science*, 4973:129–140.

Moore, J.H, Barney, N., Tsai, C.T, Chiang, F.T, Gui, J., and White, B.C (2007). Symbolic modeling of epistasis. *Human Heridity*, 63(2):120–133.

Moore, J.H, Parker, J.S., Olsen, N.J, and Aune, T. (2002). Symbolic discriminant analysis of microarray data in autoimmune disease. *Genetic Epidemiology*, 23:57–69.

Perkis, Tim (1994). Stack-based genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, Florida, USA. IEEE Press.

Ritchie, M. D., Hahn, L. W., and Moore, J. H. (2003). Power of multifactor dimensionality reduction for detecting gene-gene interactions in the presence of genotyping error, phenocopy, and genetic heterogeneity. *Genetic Epidemiology*, 24:150–157.

Ritchie, M. D., Hahn, L. W., Roodi, N., Bailey, L. R., Dupont, W. D., Parl, F. F., and Moore, J. H. (2001). Multifactor dimensionality reduction reveals high-order interactions among estrogen metabolism genes in sporadic breast cancer. *American Journal of Human Genetics*, 69:138–147.

Sempere, LF, Martinez, P, Cole, C, Baguna, J, and Peterson, KJ (2007). Phylogenetic distribution of micrornas supports the basal position of acoel flatworms and the polyphyly of platyhelminthes. *Evolution and Development*, 9(5):409–415.

Spector, Lee (2001). Autoconstructive evolution: Push, PushGP, and Pushpop. In Spector, Lee, Goodman, Erik D., Wu, Annie, Langdon, W. B., Voigt, Hans-Michael, Gen, Mitsuo, Sen, Sandip, Dorigo, Marco, Pezeshk, Shahram, Garzon, Max H., and Burke, Edmund, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 137–146, San Francisco, California, USA. Morgan Kaufmann.

Spector, Lee (2003). An essay concerning human understanding of genetic programming. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practice*, chapter 2, pages 11–24. Kluwer.

Spector, Lee and Robinson, Alan (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

Thornton-Wells, T. A., Moore, J. H., and Haines, J. L. (2004). Genetics, statistics and human disease: Analytical retooling for complexity. *Trends in Genetics*, 20:640–7.

Vladislavleva, Ekaterina, Smits, Guido, and Kotanchek, Mark (2007). Soft evolution of robust regression models. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 2, pages 13–32. Springer, Ann Arbor.

Yu, T., Riolo, R., and Worzel, B. (Eds.) (2006). *Genetic Programming Theory and Practice III*. Springer.

Chapter 10

EXPLOITING TRUSTABLE MODELS VIA PARETO GP FOR TARGETED DATA COLLECTION

Mark Kotanchek[1], Guido Smits[2] and Ekaterina Vladislavleva[3]
[1]*Evolved Analytics L.L.C., Midland, MI, USA;* [2]*Dow Benelux B.V., Terneuzen, the Netherlands;*
[3]*University of Antwerp, Antwerp, Belgium.*

Abstract Multi-objective symbolic regression has some unique capabilities in that it can focus on driving variables in the supplied data, produce white-box expressions which are human interpretable and can produce trust metrics on model quality via the assembly of ensembles of diverse but high-quality models. Exploiting these features allows adaptive data collection and iterative modeling. This is effectively an adaptive DOE (design-of-experiments) approach which is useful in a variety of scenarios: (1) optimization of an operating chemical plant whose production cannot be interrupted, (2) high-throughput empirical research wherein many variables of unknown significance are present, (3) emulator development wherein accurate surrogates for complex systems can be deployed for on-line optimization and human understanding and, even, (4) maximizing model validity for model-based decision making (e.g., development of models to assist in choosing which law school candidates should be admitted). In each of these scenarios we desire quality models; however, data collection needs to be carefully targeted for maximum efficiency.

The approach presented involves developing model, ensembles from a baseline data set. These model ensembles can be used to (a) identify potential optima in the system response and (b) identify regions of parameter space where the predictions are suspect (since in these regions the constituent models diverge). Collecting data in each of these regions helps to ensure that the target system is accurately modeled and the system optimum is accurately located.

The proposed concepts are demonstrated against a variety of systems to illustrate the key concepts and efficacy in systems ranging from low-dimensional to high-dimensional.

Keywords: symbolic regression, white box, active learning, model ensembles, trust metrics

1. Introduction

Symbolic regression is the most common industrial application of genetic programming and, essentially, involves discovering model expressions for supplied data. Symbolic regression differs from the conventional regression techniques in that a model structure is not defined *a priori*; therefore, instead of simply determining the appropriate coefficient values, symbolic regression searches for both the model structure *and* the coefficients. Because an infinite number of models can fit finite data sets (and much more if we allow for approximate fits), we are not likely to discover the exact physical model; however, we can often discover a "good enough" model which is both sufficiently accurate and suitable for human insight as well as operational use.

Searching an infinite design space for appropriate models is computationally intensive. Fortunately, recent algorithmic advances have improved the search efficiency by many orders-of-magnitude relative to the original GP algorithms as well as avoided premature lock-in on candidate solutions. The net effect is that models can be discovered in reasonable amounts of time which are competitive with other nonlinear techniques (e.g., neural networks) in terms of model accuracy with additional advantages because of the ability to automatically identify and focus on significant variables even if the inputs are correlated or coupled.

Since the model search process lets the data determine the model, diverse model structures are typically discovered which are competitive in terms of the search objectives, e.g., model accuracy and model, complexity and/or nonlinearity. Rather than choose THE model from the developed candidates, we can form an ensemble of diverse models. This ensemble has the advantage that the constituent models will tend to agree where constrained by observed data (otherwise, the would not be classified as "good" models) yet they will tend to diverge when presented with new regions of parameter space (otherwise, they would not be "diverse" models). The ensemble, therefore, implicitly has a data-defined trust metric to complement its predictions. Knowing when NOT to trust an empirical model is a significant advantage when the targeted system can be operated outside of the data space used for model development or if the system is subject to fundamental changes in behavior.

In summary, the modern multi-objective symbolic regression algorithms allow **efficient** search of **diverse** model structures for **accurate** and **parsimonious** models amenable to human **interpretation** which are automatically focused on the **driving variables** in the supplied data. These diverse models from those generated may be assembled into **ensembles** which have an **implicit trust metric** based upon the agreement or disagreement of the constituent models.

The last three of these are unique and powerful advantages of symbolic regression. In this chapter, we exploit these features to adaptively focus data collection when acquiring data is difficult or expensive. In this situation, a design-of-experiments approach founded upon linear statistics is often not practical or desirable. Instead, based upon an initial set of information, we want to collect data so that we (a) minimize the uncertainty in the model and/or (b) maximize our awareness and understanding of the system response optimum. Depending upon the application, one or the other of these objectives may be emphasized.

We very briefly review the multi-objective GP algorithms used and the ensemble definition process which produces the trustable empirical models. However, the main focus is the exploitation of the trust metric to guide data collection to reduce model uncertainty, illustrated on two examples.

2. A brief introduction to trustable symbolic regression

Since multi-objective GP as well as some of the other leading edge ideas are critical to achieving our goal of rapid adaptive model building, we briefly review the highlights of the modern GP approaches.

Single-objective genetic programming

For the purposes of this paper, single-objective and "classic GP" refer to the style of GP originally proposed by Koza and coupled with the parameters which the community has evolved to over the subsequent years: (1) single-objective use to judge model quality – typically, a least-squared-error or correlation in the case of symbolic regression, (2) some sort of parsimony pressure function to tweak the model fitness to reward simplicity, (3) large populations of models, (4) small single-winner tournaments to determine breeding rights, (5) massive CPU investments in model building.

This approach works; however, it doesn't work as well as it could. Bloat (inappropriately large expressions featuring minimal incremental accuracy gains) is a consistent problem both because bloated models tend to be difficult to interpret and fragile when faced with new data and because such expressions take longer to evaluate – which slows down the evolutionary model building. In addition to slow discovery of bloated models, we were still faced with the problem of model selection (which of the myriad models should be used. Furthermore, the accuracy of the developed models were typically good-but-not-great – lagging in accuracy behind other nonlinear techniques such as neural networks or support vector regression.

We should note that the use of Pareto tournaments (Kotanchek et al., 2006) can easily convert single-objective GP into a multi-objective variant.

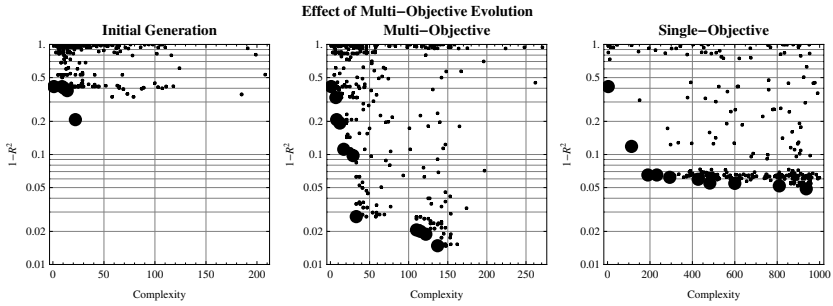


Figure 10-1. Here we show the effect of multi-objective selection on the fitness landscape for a short (60 second) evolution of a simple expression, $z = \frac{x^3+2.3}{1.2+y^2} + \text{random}[-0.2, 0.2]$. The Pareto front (optimal) models are denoted by large black disks with other models shown as smaller gray dots. Note the appearance in the knee of the Pareto front; models in this region typically represent the best trade-off of expression complexity and accuracy. For comparison, the results from a single-objective (using tournaments of size 3) symbolic regression are also shown; the conventional single-objective search is not producing the accurate and parsimonious expressions which we prefer for real-world model deployment.

Multi-objective GP

Moving to a multi-objective framework is much like an Alexander the Great solution to the Gordian Knot – with one decisive stroke it addresses many of the classic GP problems. This generates models which not only have accuracies as good or better than their neural net brethren but have additional unique and very valuable features. The key idea is that, rather than tweaking the accuracy metric to reward model simplicity (parsimony) – which may be difficult since such implies that we *a priori* know the appropriate model complexity – we *simultaneously* reward both simplicity and accuracy and choose the appropriate trade-off between expression accuracy and complexity *after* the modeling.

Complexity can be defined in a multitude of ways reflecting the structure, dimensionality, or non-linearity of models. For the experiments in this chapter we define complexity of tree-based GP model as a combination of the expressional complexity (Smits and Kotanchek, 2004) and a non-linearity measure quantifying the smoothness of the associated response surface (Vladislavleva et al., 2008).

By explicitly performing multi-objective model selection, we let the data tell us the appropriate complexity levels. This behavior is illustrated in Figure 10-1 where the fitness distributions of the initial and final model populations show that the evolutionary effort is directed at simultaneously improving both of the objectives. For comparison, classic single-objective result is also shown.

Observing a distribution of models on the fitness landscape, we can identify a *Pareto front* – a set of optimal trade-offs in model complexity and model's

prediction accuracy. From a multi-objective viewpoint, these models are ALL optimal; hence, we award propagation rights based upon model position relative to the Pareto front. Over generations, this has the effect of driving the Pareto front towards smaller model error *and* smaller complexity. An important feature of the evolved model front is the typical appearance of a knee which is the "sweet spot" from which we would typically select models in post-processing. (There are a variety of possible multi-objective selection strategies; the Pareto Tournament strategy described in (Kotanchek et al., 2006) has proven itself to be both efficient and effective. This strategy is quite simple: select random subsets of models, e.g. 30 models, and any models on the Pareto front win breeding rights. The amount of selectivity can be controlled by adjusting the subset size.)

Although a variety of fitness metrics can be used, we tend to use $(1 - R^2)$ as the accuracy metric since it is scale-invariant and, therefore, removes the requirement that model scale and translation factors be discovered at the same time as the model structure. In other words, a scale-invariant fitness metric dramatically improves the efficiency of the model search. For a complexity metric, we often use the sum of the sum of the number of nodes in the genome tree structure. For the purposes of this chapter we enrich the notion of model complexity with interval arithmetic and non-linearity measure to not only encourage structural but also a behavioral diversity and similarity of models in trustable model ensembles.

It is important to emphasize that adopting a multi-objective framework results in huge gains in the symbolic regression efficiency since the evolutionary effort is devoted towards developing interesting models rather than the evaluation of bloated expressions which, in addition to not being of practical interest, are computationally intensive to evaluate.

Other extensions

Efficient model development is a critical real-world application of symbolic regression. There are other complementary techniques which are useful in achieving that objective. Niching to allow models to develop before being forced to compete against established and mature models helps to sustain constant innovation in searching the model space. To some extent, the multi-objective algorithms naturally do this; however geographic separations like simple geographies (Spector and Klein, 2005) and islands (Koza, 1992), and also age-based niching (Hornby, 2006) have been proven to be effective.

Data balancing also tends to be important in model-building from industrial data sets where some regions of parameter space are often over-represented and, therefore, can skew the model selection towards higher performance in those (local) regions rather than targeting global generalization.

Of course, for any evolutionary strategy there are myriad implementation subtleties which, in aggregate, can yield substantial gains in model development efficiency.

The definition and implications of diverse model ensembles

There are an infinite number of models which will fit a finite data set *exactly*. When we relax this objective to include models which have a good *approximate* fit – something that is appropriate for noisy data sets – the space of possible model structures gets even larger. Our preference for parsimonious expressions reduces this somewhat; however, even if we restrict our attention to models near the knee of the Pareto front, we will still find diverse model structures which are comparable from an accuracy and parsimony perspective.

Diverse – but accurate – models have two very attractive characteristics:

- their predictions will tend to *agree* where they have common experience – i.e. when the new data is in the region covered by the training data used in the model development and
- the predictions will tend to *disagree* when they are confronted with new regions of parameter space or the targeted system has undergone some sort of fundamental change.

Knowing when NOT to trust a model can be very valuable! There is, however, the problem of identifying diverse models since we need to define "diverse". Ideally, the selected models should have different model structures, similar non-linear behavior, diverse error residuals, and (possibly) different constituent variables. Despite considerable efforts of the researchers in EC (Burke et al., 2004) an efficient characterization of structural and behavioral diversity of models and selection of a small set of appropriate candidates is difficult. A simplest expedient strategy which tends to produce the desired diversity is to simply select models whose error residuals for the available data are uncorrelated. (For large noisy data sets, the good models will tend to follow similar trajectories through the "fuzzy" observed response surface so we may need to relax the definition of "uncorrelated" under those conditions beyond what a statistician would normally define to be uncorrelated.)

The constituent models within an ensemble should be of comparable complexity and accuracy. The multi-objective symbolic regression, therefore, helps to satisfy this requirement as well as greatly improving the efficiency of the model development. However, because we want diversity in the ensemble, we prefer a loose rather than tight focus on the knee of the Pareto front. Once we have defined the "box region" of the fitness landscape, we will often restrict the candidate models to some fraction within the box closest to the Pareto front, e.g., 50%, which has the effect of selecting from a strip along the Pareto front.

Assuming that we have executed a number of independent evolutions searching for appropriate models, we could have tens of thousands of candidate models to choose between in assembling an ensemble of diverse models (as defined by the relative behavior of their error residuals). Brute force calculation of each model's correlation with all other models scales as N^2 and is not a practical selection strategy. Fortunately, a divide-and-conquer approach of partitioning the models into random subsets and letting the diverse models from each subset compete is computationally tractable and will produce an ensemble close to that generated by the brute-force strategy.

Using an ensemble requires defining a prediction function and a trust metric. For prediction we will typically average models near the median of the model predictions at any given instant. Since the median model will change as we move about parameter space, we average multiple models near the median to give a smoother prediction surface while still being robust with respect to outliers. Depending upon the number of models in the ensemble, we will either use the overall model prediction spread or some summary statistic of the constituent model predictions.

Summary on trustable symbolic regression

From the user perspective, using traditional empirical models can be nerve-wracking because of the implicit assumption that the future resembles the past and that we will not attempt to use the model in new operating regions. This is analogous to driving a car with the windshield painted over and only using the rear windows for guidance. Trustable symbolic regression cannot totally clean the window; however, it can give early warning that a curve is being encountered and the user should proceed cautiously.

The use of a multi-objective framework has provided **focus** (by automatically selecting driving variables and letting the data determine the appropriate levels of model complexity) and **insight** (since the model expressions are white boxes subject to human interpretation). Ensembles of diverse but comparable performance and complexity models leads to a **trust** metric. Each of these is a unique and valuable characteristic; their aggregate has made trustable symbolic regression the dominate technology in our industrial data modeling applications.

In the ensuing, we exploit these characteristics to guide data collection – a process we call adaptive design-of-experiments (DOE).

3. Scenarios and motivations for adaptive DOE

Adaptive DOE would be desirable when data collection is expensive or difficult. In this section we outline four scenarios which meet these criteria and would benefit from adaptive DOE despite differing considerably in their goals and emphasis: (1) operational optimization, (2) high-throughput research, (3)

emulator development and (4) law school applicant selection. The summary is that the trust metric allows us to know when the model prediction should be treated with suspicion. It also allows us to target data collection to extend the operational range of the modeling

4. The Adaptive DOE Strategy

In this section, we outline the proposed adaptive DOE strategy as well as briefly compare this approach to one founded in classical statistics.

The basic adaptive DOE strategy

The flow of the proposed strategy is outlined below.

1st: Build model ensembles from the available data. The first step is to build trustable symbolic regression models using available data. The starting data can either be from a designed experiment or historical (in which case, it should be balanced and, if necessary, pruned so that the data collected in the future will be able to influence the model selection). This gives us an initial ensemble of diverse models.

2nd: Identify the location of potential extrema. Assuming that we have a goal to optimize the system response (e.g., the first and second scenario above), we now know where we might want to collect additional data to understand the response behavior near these points as well as confirm the presence of the potential optima. If our goal is simply to maximize the trustworthiness of the model, we could possibly skip this step; however, the identified extrema will move around quite dramatically in the early phases of of model development so the diversity introduced via these points is generally desirable.

3rd: Identify locations of maximal model uncertainty. This step identifies regions where we are NOT confident about the response model validity and, as a result, where we should collect additional data to improve the quality of the model (important for the third and fourth scenarios above) and either confirm or deny the presence of local or global optima (e.g., the first two scenarios). Obviously, if our goal is developing an emulator that is valid over a wide operating range, this would be very important.

4th: Collect new data. Run the experiments to collect the appropriate data. Obviously, this needs to consider physical and safety constraints.

5th: Build new models and repeat. Using the additional information, we would then commence another round of model building – possibly including data balancing and driving variable identification – and repeat this itera-

tive process until either a sufficient model is achieved or other corporate limits are exceeded.

Comparison with classical DOE

Although the above four steps may seem somewhat simplistic, let us contrast them with an approach based on linear statistics. The sequence in this case would be:

- assume which variables are important;
- assume variables are independent and uncorrelated;
- assume a linear model form (e.g., 2nd order polynomial with single cross-terms);
- design a set of experiments featuring the "corners" (assuming sufficient experiments are possible) of the design space and (possibly) some internal or edge points;
- check if the conjecture on a pre-defined model structure is valid. If not, start over with different assumptions.

This approach has problems when the (a) variable assumptions are violated, (b) the assumed (non-physical) model form is not appropriate, (c) there are too many variables (which leads to a prohibitively large number of experiments being required), (d) the corners or edges of the design space are physically unreachable, or (e) the targeted system is operational and interrupting production to conduct a designed experiment is not desirable.

Incremental learning through the iterative process of experimental design, empirical model building, and analysis of the model is the classics of the response surface methodology approach (RSM) (see (Box and Draper, 1986)). Application of the generic idea of targeted collection through ensemble disagreement can be traced back to early 90s, and was initially used for classification through ensembles of neural networks, and so-called query algorithms (Krogh and Vedelsby, 1995; Seung et al., 1992; Freund et al., 1993). Later this framework was extended to active learning for regression type problems also, although most applications are still in classification.

We would like to emphasize, that ensembles in this application of trustable symbolic regression are created for estimating uncertainty of prediction, and ensemble models are constrained to agree on all available data points and be diverse, in a sense that they disagree in predictions everywhere but in the data records. This approach is different from boosting, which combines models that are accurate on different subsets of data for improving the general prediction accuracy, and understands the model diversity in terms of diversity of the information captured by ensemble members.

Trustable symbolic regression will automatically focus on important variables and can naturally handle correlated variables. It also lets the data define the appropriate model structure; even though this structure might not match a first-principles form, it still tends to be physically appropriate. An additional benefit is that the model development and extracting insight of the resulting models is much easier with white-box models from symbolic regression than it is with models obtained through other regression methods (like linear regression, kriging, neural networks, or support vector machines).

Prior Work

There are many real-world problems where the classical DOE assumptions are violated. Unfortunately, the need for *a priori* definition of the model structure means that the workarounds that have been developed requiring intensive effort and intuition of the experts to achieve success.

One of the inspirations for our work in adaptive DOE was Stephan Asprey's work in model-based design of experiments wherein he was designing chemical system experiments to discriminate between candidate kinetics models. Unlike symbolic regression, he would define candidate models based upon first-principles; however, like what we propose, he would collect data which would maximize the information content of additional data.

The key ingredient to the trustable symbolic regression and, subsequently, to adaptive DOE is the ability to generate *diverse* data-driven models of *comparable structural and behavioral complexity*.

5. Practical Considerations

Reality has a way of intruding upon even the most beautiful theoretical edifices. The approach we outline is remarkably robust; however, there are some real-world factors which often have to be accommodated. Of course, we assume that the measurements are accurate and the process is stable (truth shouldn't be variable). Additionally, there is an implicit assumption that a simple – albeit, possibly nonlinear – model is appropriate so that we are not exploring a fractal or chaotic response surface.

Finding the needle-in-the-haystack. An implicit assumption in the proposed adaptive DOE scheme is that we can *see* the effect of the driving variables – in other words, that the feature of interest is global rather than local. If this isn't true, then we must rely upon luck to stumble across a sample that will provide a cue that this region is special and should be investigated further.

The proposed adaptive DOE scheme is less restrictive that it would be if *a priori* model structures were imposed and the sampling were not done adaptively. However, it is not a panacea that can easily handle deceptive systems.

How many initial samples and how should they be distributed? Traditional design-of-experiments attempt to initially cover the parameter space. The distribution of sampling points depends upon the assumed model structure subject to feasibility limitations on the number of experiments which can be conducted. In many circumstances, the curse of dimensionality means that the desired experiments cannot be run.

The classic approach to work around these practical limitations might be to use a Placket-Burman DOE to attempt to identify significant variables (based upon an extremely simple model) with this followed up by a fractional factorial design. Our approach is similar – to try to maximize coverage of the parameter space within a constraint of relatively few initial data points. Since this approach will place the initial points on the convex hull of the parameter space, we also like to include the central point to provide an interior point for the model building. If the problem is deceptive, we may want to initially overweight the interior region of the parameter space with the expectation that the perimeter will be naturally explored as the adaptive DOE seeks to drive uncertainty out of the evolving model.

Rejecting pathological models. In a high-dimensional space with sparse data records, it is easily possible to develop models which fit the available data well but also have pathologies within the data variable ranges. Since searching for such pathologies in the models is computationally intensive, we use interval arithmetic (see e.g. (Keijzer, 2003)) to efficiently identify and eliminate models with potential pathologies from the evolutionary process. Due to the nature of interval arithmetic, this filtering is conservative and will reject some structures which, in fact, are not pathological. However, the overall robustness benefits make accepting the false rejections desirable.

Driving variable selection. In applications such as high-throughput research or emulator development, we are often faced with many candidate input variables. The classic way to deal with too many variables is to impose severe restrictions on the allowable model forms. To a large extent, the multi-objective symbolic regression relaxes this restriction in that we can let the evolutionary process sort through and prioritize variables for investigation based upon the available data (see (Smits et al., 2005)). We will typically place a fairly loose upper limit on the number of constituent variables or model complexity based upon the number of records used in the modeling – under an assumption that some sort of data balancing has been performed so that the effect of duplicate or near-duplicate records has been mitigated.

What to do with unmodeled variables? When working with large numbers of variables and small data sets, it is easy to encounter spurious correlations

which result in these variables being included in the model ensemble and the true drivers being excluded. For this reason, we do not want to prematurely focus on input variables – especially when the adaptive DOE process will eventually collect data which will reject the false drivers and identify the true ones. Variables which are included in the ensemble can have their values adjusted according to the search of the model response or uncertainty; the question is what to do with the ones that are not included? Our current approach is simply to randomly sample those variables within their parameter range.

Collecting multiple data records per round of adaptive DOE. Depending upon the system being modeled, we may want to do more than simply collect data at the point of maximum model uncertainty or at the extrema. Generally the developed model is multi-modal so rather than trying to find the global maxima of these characteristics, we can start from random points in parameter space and use local rather than global extrema as additional possible sampling points.

Reducing repeated sampling at the corners. The behavior of the developed models are less constrained at the boundaries of parameter space since the constraint on the function behavior is single-sided. This freedom to deviate is even more at the corners. As a result, the automated algorithm can often repeatedly identify such locations as points were more data would be good – even though we have already sampled that location! In such situations, we initiate a selection of local searches from random starting points and use the most interesting points identified which have not been previously sampled.

Building on previous results. For computational efficiency in the demonstrations, as each new set of data points would be added to the collection, we would initialize the models supplied to the next round with models from the current round. The attraction of such an strategy is that the initial models for each round of adaptive DOE should be relatively good (at least in the later stages) and the symbolic regression search does not need to spend effort re-discovering driving variables and quality model structures.

Although the use of prior models is useful in the latter stages of model refinement, in the early stages each incremental data point is likely to shatter most of the models which were evaluated against the prior data. Hence, we will also want to have some symbolic regressions starting from scratch since the previous efforts may have eliminated true driving variables from the population. The summary is that we want to have multiple independent evolutions running at each round of the adaptive DOE process with some of them using previously developed models as an initial population and others starting with random models to avoid this bias.

Identifying the knee of the Pareto front. Experience has shown that the "knee of the Pareto front" is an appropriate region from which to select models for our ensembles. Identifying the inflection point representing the best trade-off between expression complexity and accuracy can be difficult – and the perceived knee location can shift depending upon whether linear or log scaling is used to display the error axis.

Assuming the knee has been identified, we have the problem of defining the region from which the ensemble models should be drawn. In the interests of diversity and towards a goal of detecting when the prediction should not be trusted, we want to include both (a) less-accurate-but-simpler models under the premise that they capture more of the lower-order global behavior and are less prone to chasing (possibly spurious) nuances in the response behavior and (b) more-accurate-but-more-complex models with the expectation that these models will be more "twitchy" than their less complex compatriots and, therefore, quicker to detect that the ensemble is entering novel territory or that the underlying system has changed.

In principle, we want a human in the loop to identify these "interesting" models; however, for the purposes of the examples in this chapter, we adopted a very simple and reasonably robust approach:

- Identify *a priori* a region of the the location of good enough models. For this chapter, we defined this as models with a complexity of less than 400 and an $R^2 \geq 0.9$ (implying $1 - R^2 \leq 0.1$ which is our error metric).

- Of the models within this boxed region, identify the 90% quantiles for both performance metrics. This defines a region of interest that floats with the varying model quality.

- Select the 50% of the (unique) models within this box which are closest to the Pareto front. If this process does not identify at least 30 models, iteratively increase the allowable model error by 10% until at least this number of models is selected.

From this population of models, build an ensemble of uncorrelated models from (a) the overall set of candidates and (b) the Pareto front of the candidates.

The resulting ensemble is then used to identify regions of parameter space which should be explored.

Handling quantized variable spaces & historical data sets. Sometimes the data is quantized and, as a result, we have restrictions as to where additional data can be collected. In this case, our only recourse is to use the "closest" data point where now the choice of an appropriate distance metric is significant.

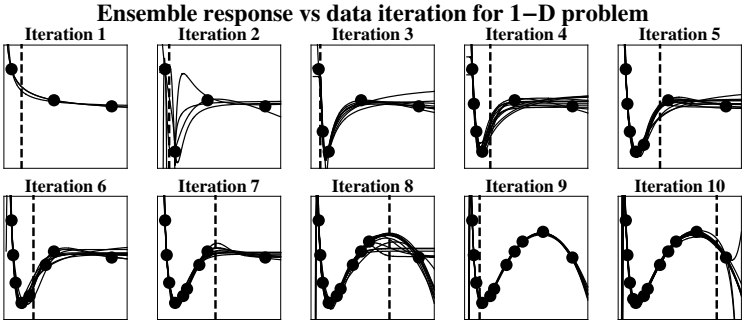


Figure 10-2. This shows a typical adaptive DOE sequence for the 1-D example. The data points used to build the model are denoted by solid disks and the predictions of ensemble models for the available data by the solid line. For each round of experimentation. The location of the next sample point is indicated by a vertical dashed line. This is the point where divergence of the models embedded within the ensemble is maximized. (The divergence is computed for the interval $[0.5, 7]$, while the solutions are plotted on the interval $[0, 8]$.

For large historical data sets, we might adopt a similar approach and starting from a small initial set of samples iteratively mine the historical data to build models and select variables.

"Box" sampling vs. operational constraints. It may not be physically possible or safely possible to reach all possible variable combinations – especially in high-dimensional systems such as an in-production plant. In this case, we may want to gradually expand the search space from the known (presumably safe) operating points. In this case, we may want to choose our additional sample points such that they are within the convex hull of the previously sampled data or within some limited distance from a previously sampled data point. Although this is a significant issue, we do not address it further in this chapter since appropriate strategies are very problem-specific.

6. Demonstration : 1-D function

In this example we attempt to discover a very simple function which is a linear combination of a decaying exponential, an increasing square root and a negative parabolic over the range $x \in [0.5, 7]$: $\frac{3}{x} + 5\sqrt{x} - 0.1x^2$.

Starting from three data points sampled from this function we apply the adaptive DOE strategy in 10 iterations and add one new point per iteration, identified as a point of maximal ensemble disagreement. The response surfaces of ensemble models at each iteration are plotted in Figure 10-2 with extrapolation. At each iteration, the developed ensemble would have to be considered as an excellent set of models – it does an excellent job of fitting the observed data –

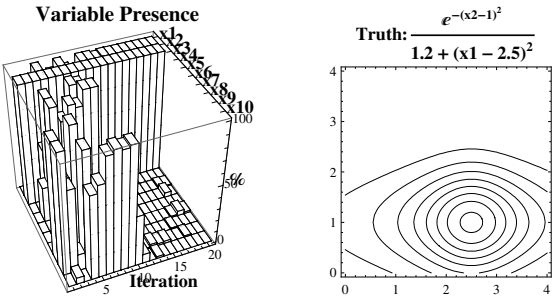


Figure 10-3. The bar chart plot at the left demonstrates the distribution of variables in the models located at the Pareto front of the GP archive per each DOE iteration. The contour plot at the right corresponds to the original response surface (only two out of 10 input variables are significant). For this adaptive DOE run we started with a base set of 12 sample points randomly distributed in a 10-D space. For each iteration, three new samples were collected: (1) at the point of maximum ensemble disagreement, (2) the model maximum and (3) the model minimum. Using this additional targeted data collection, the adaptive DOE process was able to quickly isolate on the true driving variables (from iteration 10 on) and build high-quality models.

despite the fact that it doesn't do all that good of a job (in the early stages) of fitting the true function.

Since in real-life we only have the data, the ability of the data to tell us both a working hypothesis for the system model and where additional data would be useful is rather impressive. Even though by the time we get to the tenth iteration (12 data points) fairly complete coverage of the parameter space has been achieved, we still have diversity in the behaviour of the ensemble models (see response curves under extrapolation in Figure 10-2).

7. Demonstration : 2-D function embedded in 10-D

In this example, we use the Kotanchek function, but embedded in a 10-D hypercube, $\mathbf{x} \in [0,4]^{10}$. The Kotanchek function is somewhat deceptive in that it is flat over a large fraction of parameter space. It is also difficult to model since the expression structure is quite complicated and we are not using either exponentials or powers as building blocks (other than square-root and square) during the modeling. Starting from 12 data points distributed within this hyper-space, we want to quickly identify and isolate on the true driving variables and build a good model. For each DOE round, we build an ensemble from the "good" models found at the knee of the Pareto front using the method discussed previously. The ensemble is comprised of models whose error residuals are jointly uncorrelated. From the ensemble, we identify the predicted maximum and minimum points as well as the location of maximum disagreement of the models embedded within the ensemble. We then go and

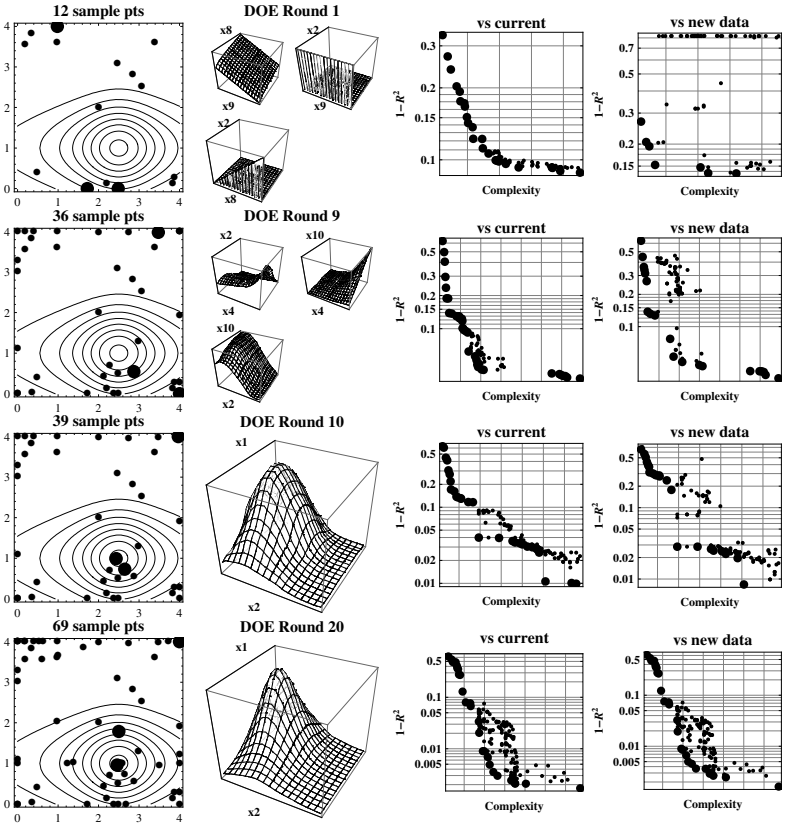


Figure 10-4. Each row of the plots presents a particular round of the adaptive DOE. The three new points sampled at a corresponding iteration are depicted in bold in the contour plots in the first column (note, that projections of the 10-D space onto the subspace of the two driving variables are presented). In the second column the best models of the iteration are presented. Note that up until round 10, the system could not discover the true driving variables $x1$ and $x2$. The plots in the third column represent GP solutions in the objective space of model complexity and model accuracy on the available data (with Pareto front models emphasized in bold). Plots in the fourth column present the same GP models, but with accuracy evaluated on the data, extended with three new data samples identified with the adaptive DOE strategy. Note that up until round 20, new sample points tend to significantly distort the accuracy of solutions.

collect data at those points and use the augmented information to develop a next round of models. (We should also note that we have made the model search more difficult via a limited function set (plus, subtract, times, divide, square root and square) and we are not including e as an available constant. We performed six independent evolutions per adaptive DOE round with each round allocated between 15 seconds and 2 minutes of CPU time depending upon the data set size.)

Figure 10-3 shows the changing composition of the models at the knee of the Pareto front with each DoE round. As might be expected with many input variables and limited data, it is easily possible to construct apparently accurate models from spurious relationships. As a result, in the early stages the evolutionary search may home in on erroneous model structures or variables which fit the observed behavior quite well.

We illustrate this in Figure 10-4 by showing four stages of the adaptive DOE process at iterations 1, 9, 10, and 20. In the early stages of the adaptive DOE strategy we have too many variables and not enough data so it is difficult for the multi-objective symbolic regression to identify the true driving variables. However, by using targeted data sampling, it is possible to automatically test the importance of input variables and focus on the true drivers using relatively few additional data records. This is illustrated in the last columns of Figure 10-4, where the model fitness distribution of "good" models is shattered by the inclusion of new data into the fitness evaluation, so, the good models chose where to collect data to best separate the pretenders from the contenders.

One of the beautiful features of the adaptive DOE process is that the new data samples will quickly expose these models as flawed. This is illustrated in Figure 10-4 wherein the data collected after the 1st and 9th round of model building destroys the illusion of high-quality models. Conversely a few rounds later, the incremental data doesn't expose fundamental flaws in the models and, instead, is used to refine the model quality and to further isolate on the driving variables.

Summary and Conclusions

Trustable symbolic regression has some huge advantages for industrial applications. In this chapter we propose exploiting those unique advantages to guide data collection. Active data collection obtained through ensembles of global white-box GP models could have significant advantages to efficiently develop models of high-dimensional systems where the driving variables are unknown and there are significant costs and constraints on the data collection.

References

Box, George E P and Draper, Norman R (1986). *Empirical model-building and response surfaces*. John Wiley & Sons, Inc., New York, NY, USA.

Burke, Edmund K., Gustafson, Steven, and Kendall, Graham (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62.

Freund, Yoav, Seung, H. Sebastian, Shamir, Eli, and Tishby, Naftali (1993). Information, prediction, and query by committee. In *Advances in Neural*

Information Processing Systems 5, [NIPS Conference], pages 483–490, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Hornby, Gregory S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 815–822, Seattle, Washington, USA. ACM Press.

Keijzer, Maarten (2003). Improving symbolic regression with interval arithmetic and linear scaling. In Ryan, Conor, Soule, Terence, Keijzer, Maarten, Tsang, Edward, Poli, Riccardo, and Costa, Ernesto, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 70–82, Essex. Springer-Verlag.

Kotanchek, Mark, Smits, Guido, and Vladislavleva, Ekaterina (2006). Pursuing the pareto paradigm tournaments, algorithm variations & ordinal optimization. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12, pages 167–186. Springer, Ann Arbor.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Krogh, Anders and Vedelsby, Jesper (1995). Neural network ensembles, cross validation, and active learning. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238, Cambridge, MA, USA. The MIT Press.

Seung, H. S., Opper, Manfred, and Sompolinsky, Haim (1992). Query by committee. In *Computational Learning Theory*, pages 287–294.

Smits, Guido, Kordon, Arthur, Vladislavleva, Katherine, Jordaan, Elsa, and Kotanchek, Mark (2005). Variable selection in industrial datasets using pareto genetic programming. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 6, pages 79–92. Springer, Ann Arbor.

Smits, Guido and Kotanchek, Mark (2004). Pareto-front exploitation in symbolic regression. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 17, pages 283–299. Springer, Ann Arbor.

Spector, Lee and Klein, Jon (2005). Trivial geography in genetic programming. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 8, pages 109–123. Springer, Ann Arbor.

Vladislavleva, Ekaterina, Smits, Guido, and den Hertog, Dick (2008). Order of non-linearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, In Press.

Chapter 11

EVOLVING EFFECTIVE INCREMENTAL SOLVERS FOR SAT WITH A HYPER-HEURISTIC FRAMEWORK BASED ON GENETIC PROGRAMMING

Mohamed Bader-El-Den[1] and Riccardo Poli[1]
[1]*Department of Computing and Electronic Systems, University of Essex.*

Abstract Hyper-heuristics could simply be defined as heuristics to choose other heuristics. In other words, they are methods for combining existing heuristics to generate new ones. In this paper, we use a grammar-based genetic programming hyper-heuristic framework. The framework is used for evolving effective incremental solvers for SAT. The evolved heuristics perform very well against well-known local search heuristics on a variety of benchmark SAT problems.

Keywords: hyper-heuristic, Inc, SAT, heuristics.

1. Introduction

Heuristic methods have contributed to the solution of many combinatorial optimisation problems such as bin packing, the travelling salesman problem (TSP), graph colouring, and the satisfiability problem (SAT). The performance of heuristics on a problem varies from instance to instance. Also, even on the same instance, randomised heuristics may be able to provide good solutions on one occasion, and bad on another. Hyper-heuristics (HHs) aim at providing a more robust approach raising the level of generality at which optimisation methods operate. They can be defined as "heuristics to choose heuristics" (Burke et al., 2003a). The main idea is to make use of different heuristic during the search for a solution.

SAT is one of the most studied combinatorial optimisation problems, and the first problem proved to be NP-Complete. In this paper we will use genetic programming (GP) in a HH framework to solve SAT problems. In particular, we

use GP to evolve local search heuristics to be used within the Inc* algorithm. Inc* (Bader-El-Den and Poli, 2008) is a general algorithm that can be used in conjunction with any local search heuristic and that has the potential to substantially improve the overall performance of the heuristic.

The general idea of the algorithm is the following. Rather than attempting to directly solve a difficult problem, the algorithm dynamically chooses a simplified instance of the problem, and tries to solve it. If the instance is solved, Inc* increases the size of the instance, and repeats this process until the full size of the problem is reached. The search is not restarted when a new instance is presented to the solver. Thus, the solver is effectively and progressively biased towards areas of the search space where there is a higher chance of finding a solution to the original problem. We look at all this in more detail below.

SAT problem

The target in SAT is to determine whether it is possible to set the variables of a given Boolean expression in such a way to make the expression true. The expression is said to be satisfiable if such an assignment exists. If the expression is satisfiable, we often want to know the assignment that satisfies it. The expression is typically represented in Conjunctive Normal Form (CNF), i.e., as a conjunction of clauses, where each clause is a disjunction of variables or negated variables.

There are many algorithms for solving SAT. Incomplete algorithms attempt to guess an assignment that satisfies a formula. So, if they fail, one does not know whether that's because the formula is unsatisfiable or simply because the algorithm did not run for long enough. Complete algorithms, instead, effectively *prove* whether a formula is satisfiable or not. So, their response is conclusive. They are in most cases based on backtracking. That is, they select a variable, assign a value to it, simplify the formula based on this value, then recursively check if the simplified formula is satisfiable. If this is the case, the original formula is satisfiable and the problem is solved. Otherwise, the same recursive check is done using the opposite truth value for the variable originally selected.

The best complete SAT solvers are instantiations of the Davis Putnam Logemann Loveland procedure (Davis et al., 1962). Incomplete algorithms are often based on local search heuristics (see next section). These algorithms can be extremely fast, but success cannot be guaranteed. On the contrary, complete algorithms guarantee success, but they computational load can be considerable, and, so, they cannot be used for large SAT instances.

Stochastic local-search heuristics. Stochastic local-search heuristics have been widely used since the early 90s for solving the SAT problem following the successes of GSAT (Selman et al., 1992). The main idea behind these heuristics

Algorithm 1

$L =$ initialise the list of variables randomly
for $i = 0$ to MaxFlips **do**
  if L satisfies formula F **then**
    return L
  end if
  select variable V using some selection heuristic
  flip V in L
end for
return no assignement satisfying F found

is to try to get an educated guess as to which variable will most likely, when flipped, give us a solution or move us one step closer to a solution. Normally, the heuristic starts by randomly initialising all the variables in the CNF formula. It then flips one variable at a time, until either a solution is reached or the maximum number of flips allowed has been exceeded. Algorithm 1 shows the general structure of a typical local-search heuristic for the SAT problem. The algorithm is normally repeatedly restarted for a certain number of times if it is not successful.

The best heuristics of this type include:

- GSAT: (Selman et al., 1992) which, at each iteration, flips the variable with the highest gain score, where the gain of a variable is the difference between the total number of satisfied clauses after flipping the variable and the current number of satisfied clauses. The gain is negative if flipping the variable reduces the total number of satisfied clauses.

- HSAT: (Gent and Walsh, 1993) In GSAT more than one variable may present the maximum gain. GSAT chooses among such variables randomly. HSAT, instead, uses a more sophisticated strategy. It selects the variable with the maximum age, where the age of a variable is the number of flips since it is was last flipped. So, the most recently flipped variable has an age of zero.

- GWSAT: (Selman and Kautz, 1993) with probability p selects a variable occurring in some unsatisfied clauses while with probability $(1-p)$ flips the variable with maximum gain as in GSAT.

- WalkSat: (Selman et al., 1994) starts by selecting one of the unsatisfied clauses C. Then it flips randomly one of the variables that will not break any of the currently satisfied clauses (leading to a "zero-damage" flip). If none of the variables in C has a "zero-damage" characteristic, WalkSat

selects with probability p the variable with the maximum score gain, with probability $(1 - p)$ a random variable in C.

- Novelty: (Hoos and Stützle, 2000) After selecting a random unsatisfied clause, Novelty flips the variable with the highest score unless it was the last variable flipped in the clause. If this is the case, with probability p the same variable is flipped, otherwise a random variable from the same clause is selected.

Evolutionary algorithms and SAT problem. Different evolutionary techniques have been applied to the SAT problem. There are two main research directions: direct evolution and evolution of heuristics.

An example of methods of the first type – direct evolution – is FlipGA which was introduced in (Marchiori and Rossi, 1999). There a genetic algorithm was used to generate offspring solutions to SAT using the standard genetic operators. However, offspring were then improved by means of local search methods. The same authors later proposed ASAP, a variant of FlipGA (Rossi et al., 2000). A good overview of other algorithms of this type is provided in (Gottlieb et al., 2002).

The second direction is to use evolutionary techniques to automatically evolve local search heuristics. A successful example of this is the CLASS system developed in (Fukunaga, 2002; Fukunaga, 2004). The process of evolving new heuristics in the CLASS system is based on five conditional branching cases (if-then-else rules) for combining heuristics. Effectively CLASS can be considered as a very special type of GP system where these rules are used instead of the standard GP operators (crossover and mutation). The evolved heuristics were competitive with a number of human-designed heuristics. However, the evolved heuristics were relatively slow. This is because the conditional branching operations used in CLASS evaluate two heuristics first and they then select the output of one to decide which variable to flip. Also, restricting evolution to use only conditional branching did not give the CLASS system enough freedom to evolve heuristics radically different from the human-designed heuristics (effectively, the evolved heuristic are made up by a number of nested heuristics). Another example of system that evolves SAT heuristics is the STAGE system introduced in (Boyan and Moore, 2000). STAGE tries to improve the local search performance by learning (online) a function that predicts the output of the heuristic based on some characteristics seen during the search.

Hyper-heuristics, GP and SAT

As we noted above, hyper-heuristics are "heuristics to choose other heuristics" (Burke et al., 2003a). A heuristic is considered as rule-of-thumb or "educated guess" that reduces the search required to find a solution. The difference

between metaheuristics and hyper-heuristics is that the former operate directly on the targeted problem search space with the goal of finding optimal or near optimal solutions. The latter, instead, operate on the heuristics search space (which consists of the heuristics used to solve the target problem). The goal then is finding or generating high-quality heuristics for a target problem, for a certain class of instances of a problem, or even for a particular instance.

There are different classes of hyper-heuristics. In one class of HH systems, the system is provided with a list of preexisting heuristics for solving a certain problem. Then the HH system tries to discover what is the best sequence of application for these heuristics for the purpose of finding a solution as shown in Figure 11-1. Different techniques have been used to build HH systems of this class. Algorithms used to achieve this include, for example: tabu search (Burke et al., 2003b), case-based reasoning (Burke et al., 2006b), genetic algorithms (Cowling et al., 2002), ant-colony systems (Silva et al., 2005), and even algorithms inspired to marriage in honey-bees (Abbass, 2001).

Another form of hyper-heuristic is one where the system produces (meta-)heuristics by specialising them from a generic template. The specialisation can take the form of one or more evolved components, which can modify the behaviour of the meta-heuristic or heuristic. This approach has given, for example, very positive results in (Poli et al., 2007) where the problem of evolving offline bin-packing heuristics was considered. There GP was used to evolve strategies to guide a fixed solver. This approach was also taken in (Bader-El-Den and Poli, 2008) where Inc* was originally proposed. This solver was applied to the SAT problem with good success. To further improve chances of success, a key element of Inc*, its strategy for adding and removing SAT clauses, was evolved using GP. (We will provide more information about this below, since the work presented in this paper relates closely to Inc*.)

A third approach used to build HH systems is to create (e.g., evolve) new heuristics by making use of the *components* of known heuristics. The process starts simply by selecting a suitable set of heuristics that are known to be useful in solving a certain problem. However, instead of directly feeding these heuristics to the HH system (as in the first type of HHs discussed above), the heuristics are first decomposed into their basic components. Different heuristics may share different basic components in their structure. However, during the decomposition process, information on how these components were connected with one another is lost. To avoid this problem, this information is captured by a grammar. So, in order to provide the HH systems with enough information on how to use components to create valid heuristics, one must first construct an appropriate grammar. Hence, in the hyper-heuristics of this third type, both the grammar and the heuristics components are given to the HH systems. The system then uses a suitable evolutionary algorithm to evolve new heuristics, Figure 11-2 shows an abstract model for the process. For example, in recent

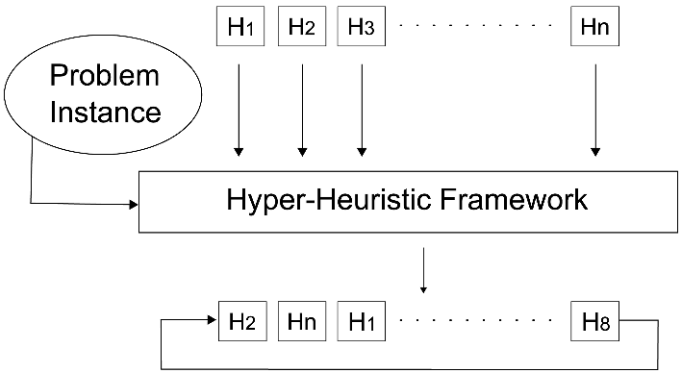


Figure 11-1. Shows an abstract structure of one of the HH types: In this type, HH tries to find the best sequence of heuristics to solve a certain problem.

work (Burke et al., 2006a) GP was successfully used to evolve new heuristics of for one-dimensional online bin packing problems. While in (Bader-El-Din and Poli, 2007) this approach was used in a system called GP-HH (for GP Hyper-Heuristic) to evolve heuristics for the SAT problem which are specialised to solve specific sets of instances of the problem. A comparison between GP-HH and other well-known evolutionary and local-search heuristics revealed that the heuristics produced by GP-HH are very competitive, being on par with some of the best-known SAT solvers.

Contributions of this Paper

The Inc* algorithm proposed in (Bader-El-Den and Poli, 2008) and briefly discussed above has been successful. There the choice of the simplified problems progressively leading to the original (hard) problem is dynamic so as to limit the chances of the algorithm getting stuck in local optima. Whenever the system finds a simplified instance too difficult, it backtracks and creates a new simplified instance. In the SAT context, the simplified problems are simply obtained by choosing subsets of the clauses in the original formula. The subset in current use is called the *clauses active list*. Depending on the result of the heuristic on this portion of the formula, the algorithm then increases or decreases the number of clauses in the active list. Naturally the choice of how many clauses to add or remove from the active list of Inc* after a success or failure is very important for the good performance of the algorithm. In (Bader-El-Den and Poli, 2008) genetic programming was used to discover good dynamic strategies to make such decisions optimally at run time. These strategies were constrained to act within the specific Inc* algorithm (which we report in Algorithm 2), which was human designed.

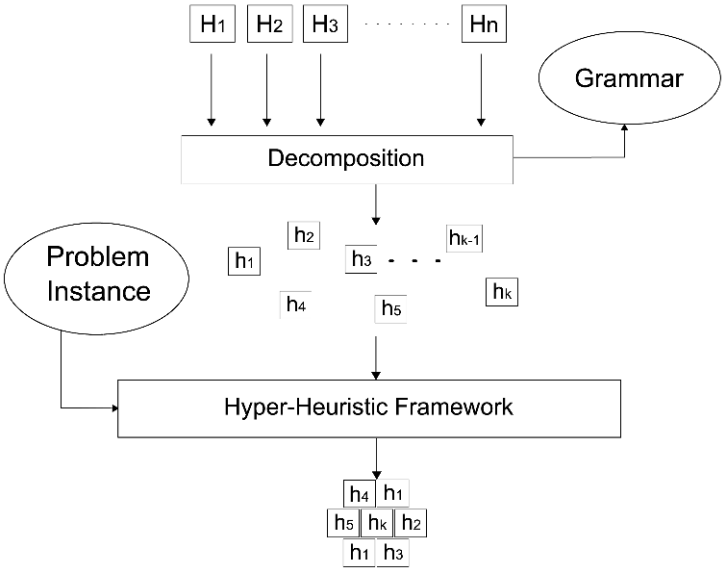


Figure 11-2. Shows an abstract structure of one of the HH types, In this type the input heuristics H are decomposed to their components h before evolving new heuristics from these components.

Encouraged by the success of Inc* and the GP-HH, in this work, we want to take Inc* one step further. We want to evolve local search SAT heuristics specifically designed to work well within Inc*, instead of just evolving the control element of Inc* which decides how many clauses to add or remove upon success or failure, which is what we did before. To do this, we make use of the grammar-based Hyper-Heuristic GP framework developed in (Bader-El-Din and Poli, 2007; Bader-El-Den and Poli, 2007). As one can easily see inspecting the local search heuristics for SAT presented above, all the heuristics share similar components, for example variable score, selection of a clause and conditional branching. The components of these heuristics plus additional primitives that are necessary to evolve more complex behaviours are used to generate the grammar for GP-HH, as described in the next section.

By giving GP-HH the freedom to design completely new Inc* searchers, we hoped to find algorithms for the solution of the SAT problem that are novel and even more powerful than Inc* or GP-HH alone.

2. GP hyper-heuristic for evolving Inc* SAT heuristics

As mentioned before, the most important step in designing a GP Hyper-Heuristic framework is constructing a grammar capable of describing heuristics for the targeted domain. The grammar shows the GP system how the elementary

Algorithm 2 Inc* approach to solving SAT problems.

```
1: L = random variable assignment
2: AC = small set of random clauses from the original problem
3: Flips = number of allowed flips at each stage
4: Flips_Total = 0 {This keeps track of the overall number of flips used}
5: Flips_Used = 0 {This keeps track of the flips used to test the active list}
6: Inc_Flip_Rate = rate of increment in the number of flips after each fail
7: repeat
8:    for i = 0 to Flips do
9:       if L satisfies formula F then
10:         return L
11:      end if
12:      select variable V from AC using some selection heuristic
13:      flip V in L
14:   end for
15:   Flips_Total = Flips_Total + Flips_Used
16:   update clause weights
17:   if L satisfies AC then
18:      if AC contains all clauses in F then
19:         return L
20:      end if
21:      AC = add more clauses to the active list
22:   else
23:      sort AC
24:      AC = remove some clauses from the active list
25:      Flips = increment allowed flips
26:   end if
27: until Flips_Total < MaxFlips
28: return no assignment satisfying F found
```

components obtained from the decomposition of previously-known heuristics could be connected to generate valid evolved heuristics. The grammar presented in this section is an extension of the one used in (Bader-El-Din and Poli, 2007; Bader-El-Den and Poli, 2007), which we modified so as to be more efficient and to better suit the Inc* framework. The grammar contains elements form the heuristic descried in the previous section.

The GP Hyper-Heuristic framework allows us to use components not used in handcrafted heuristics to see whether they can help the evolution process. This is important in our case because we are not just evolving local-search SAT heuristics but much more sophisticated heuristics for Inc*. For example, we

```
start  →   Flip v
v      →   Random l |
           MaxScr l, op |
           ScndMaxScr l, op |
           ZeroBreak l, v |
           Ifv condV, v, v |
l      →   ALL |
           UC |
           AllUC |
           Ifl condL, l, l |
condV  →   prob | size |
           NotMinAge |
           NotZeroAge |
           MinAge
condL  →   prob | size
prob   →   20 | 40 | 50 |
           70 | 90
size   →   Small | Midd |
           Large | LastAtmp
op     →   TieRandom | TieAge
```

Figure 11-3. The GP Hyper-Heuristic grammar used for evolving Inc* SAT heuristics.

added to the grammar new conditional branching conditions, based on the size of the current *clauses active list* as will be described in details later.

To construct our grammar, we classified the main components of SAT heuristics into two main groups. The first group of components, Group 1, returns *a variable* from an input list of variables (e.g., the selection of a random variable from the list or of the variable with highest gain score). The second group, Group 2, returns *a list of variables* from the CNF formula (e.g., the selection of a random unsatisfied clause which, effectively, returns a list of variables). The grammar is designed in such a way to produce functions with no side effects (i.e., we avoid using variables for passing data from a primitive to another). The aim was to reduce the constraints on the crossover and mutation operators, and to make the GP tree representing each individual simpler.

The grammar we used and its components are shown in Figure 11-3. The root of each individual (an expression in the language induced by the grammar) is the primitive `Flip`, which flips a variable `v`. The variable `v` is typically (but not always) selected from a list of variables, `l`, using appropriate primitives. There are three ways in which this can happen.

The first method is to select the variable randomly from `l`. This is done by the function `Random l`.

The second method is to choose the variable based on the score of the variables in the list. In other words, the choice depends on how many more clauses will be satisfied after flipping a variable. A positive sign of the score means that more clauses will be satisfied. A negative score is instead obtained if flipping the variable will cause fewer clauses to be satisfied. The selection based on score is done by either `MaxScr` or `ScndMaxScr`. These functions select the variable with the highest and second highest score from `l`, respectively. Both `MaxScr` and `ScndMaxScr` require a second argument, `op`, in addition to a list `l`. This argument specifies how to break ties if multiple variables have the same highest score. If `op` is `TieAge`, the tie will be broken by favouring the variable which has been flipped least recently, while `TieRandom` breaks ties randomly.

The third method for selecting a variable is through the primitive `ZeroBreak` which selects the variable that will not "unsatisfy" any of the currently satisfied clauses. If a such variable does not exist in the given list, the primitive returns its second argument `v`, which is selected by any of the previous methods.

The list of variables `l` can be selected from the CNF formula in three simple ways. The first two are directly taken from the handcrafted heuristics: `All` returns all the variables in the formula, while `UC` returns all the variables in one of randomly selected clause. The third method, `AllUC`, selects all the variables in all unsatisfied clauses. This primitive can be very useful (and, indeed, was often used by GP) in stages where the number of clauses in the *clauses active list* of Inc* is relatively small.

The grammar also includes conditional branching components (`IFV` and `IFL`). Branching components are classified on the basis of their return type. The condition `prob` means that the branching is probabilistic and depends on the value of `prob`. Also the branching can be done on other criteria like `size` of the *clauses active list*. While this is not in any of the handcrafted heuristics, we added it to our grammar to make GP able to evolve heuristics that adapt with the different stages in the Inc* algorithm, while it progressively adds and removes clauses from the active list. `LastAtmp` is *true* if the Inc* last attempt to satisfy the *clauses active list* was successful and *false* otherwise.

3. Experimental setup

In evolving Inc* SAT heuristics we used a training set including SAT problems with different numbers of variables. The problems were taken from the widely used SATLIB benchmark library. All problems were randomly generated satisfiable instances of 3-SAT. In total we used 50 instances: 10 with 100 variables, 15 with 150 variables and 25 with 250 variables. While strategies are

evolved using 50 fitness cases, the generality of best of run individuals is then evaluated on an independent test set including many more SAT instances.

The fitness $f(s)$ of an evolved strategy s was measured by running the Inc* algorithm under the control of s on all the 50 fitness cases. More precisely

$$f(s) = \sum_i \left(inc_s(i) * \frac{v(i)}{10} \right) + \frac{1}{flips(s)}$$

where $v(i)$ is the number of variables in fitness case i, $inc_s(i)$ is a flag representing whether or not running the Inc* algorithm with strategy s on fitness case i led to success (i.e., $inc_s(i) = 1$ if fitness case i is satisfied and 0 otherwise), and $flips(s)$ is the number of flips used by strategy s averaged over all fitness cases. The factor $v(i)/10$ is used to emphasise the importance of fitness cases with a larger number of variables, while the term $1/flips(s)$ is added to give a slight advantage to strategies which use fewer flips (this is very small and typically plays a role only to break symmetries in the presence of individuals that solve the same fitness cases, but with different degrees of efficiency).

The GP system initialises the population by randomly drawing nodes from the function and terminal sets. This is done uniformly at random using the GROW method, except that the selection of the function (head) $Flip$ is forced for the root node and is not allowed elsewhere. After initialisation, the population is manipulated by the following operators:

- Roulette wheel selection (proportionate selection) is used. Reselection is permitted.
- The reproduction rate is 0.1. Individuals that have not been affected by any genetic operator are not evaluated again to reduce the computation cost.
- The crossover rate is 0.8. Offspring are created using a specialised form of crossover. A random crossover point is selected in the first parent, then the grammar is used to select the crossover point from the second parent. It is randomly selected from all valid crossover points. If no point is available, the process is repeated again from the beginning until crossover is successful.
- Mutation is applied with a rate of 0.1. This is done by selecting a random node from the parent (including the root of the tree), deleting the sub-tree rooted there, and then regenerating it randomly as in the initialisation phase.
- We used a population of 500 individuals.

We have used the following parameters values for the Inc* algorithm:

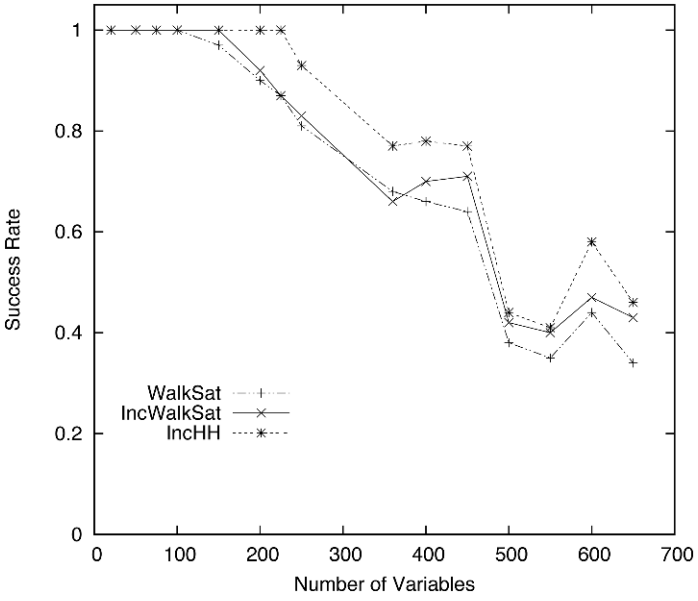


Figure 11-4. Average success rate performance of WalkSAT, IncWalkSat and IncHH.

- To start with we allow 2,000 flips for instances with more than 250 variables, while we use 100 flips for smaller instances.
- Upon failure, the number of flips is incremented by 20%.
- We allow a maximum total number of flips of 400,000 for instances with more than 250 variables, while we use 100,000 flips for smaller instances.
- The maximum number of tries is 1000 (including successful and unsuccessful attempts). This is mainly to prevent the system from getting into an infinite loop.
- We evolved Inc* SAT heuristics for one simple Inc* strategies which adds 15% clauses after each success and removes 10% after each failure.

4. Results

We start by showing a typical example of the search heuristics for Inc* evolved using the GP Hyper-Heuristics framework.

Figure 11-4 shows one of the best performing heuristics evolved for the Inc* strategy represented as a program tree. As one can see evolved heuristics are significantly more complicated than the standard heuristics we started from (e.g., GSat, WalkSat, Novelty). So, a manual analysis of how the component steps of an evolved heuristic contribute to its overall performance is difficult.

Although the initial population in GP is randomly generated and includes no handcrafted heuristics, we have noticed that most of the best evolved individuals contain patterns of instructions similar to those in hand crafted heuristics. This means that our grammar-based GP system has been able mix different heuristics patterns to create new heuristics for Inc*.

The Inc* strategy and parameter settings used during the evolution process are the same as the one used in the testing phase. Table 11-1 shows the results of a set of experiments comparing the performance of three algorithms: WalkSat alone, WalkSat with Inc* (IncWalk) and the heuristic evolved by GPHH with Inc* (IncHH). Instances with up to 250 variables were taken from SatLib (uf20 to uf250). Larger instances were taken from the benchmark set of the SAT 2007 competition. None of the test instances had been used in the GP training phase. The performance of the heuristics on an instance is the number of flips required to solve it averaged over 10 independent runs of a solver, to ensure the results are statistically meaningful. The AF column shows the average number of flips used by each heuristic in successful attempts only.

Table 11-1. Comparison between average performance of WalkSat and WalkSat with Inc* and Inc* with the evolved heuristic (IncHH) SR=success rate, BT = average back tracks , AF=average number of flips.

		WalkSat		IncWalk			IncHH		
name	clauses	SR	AF	SR	AF	BT	SR	AF	BT
uf20	91	1	104.43	1	136.32	1.13	1	98.54	0.96
uf50	218	1	673.17	1	702.52	4.25	1	723.52	3.13
uf75	325	1	1896.74	1	1970.59	8.15	1	1909.61	7.17
uf100	430	1	3747.32	1	3640.62	10.31	1	3769.42	9.07
uf150	645	0.97	15021.3	1	13526	15.44	1	6454.14	12.60
uf200	860	0.9	26639.2	0.92	27586.2	20.59	1	26340.8	19.09
uf225	960	0.87	29868.5	0.87	32258.8	21.27	1	34187.7	20.24
uf250	1065	0.81	38972.4	0.83	39303.5	25.15	0.93	39025.6	24.37
com360	1533	0.68	277062	0.66	225874	33.82	0.77	147617	31.87
com400	1704	0.66	172820	0.70	188435	31.64	0.78	191493	32.7
com450	1912	0.64	169113	0.71	190325	32.72	0.77	170459	30.36
com500	2130	0.38	271822	0.42	297683	35.72	0.45	265762	36.59
com550	2343	0.35	288379	0.40	278567	39.84	0.41	266429	38.23
com600	2556	0.44	257479	0.47	228347	36.64	0.58	297932	39.12
com650	2769	0.34	274112	0.43	285964	34.65	0.43	282352	44.30

We categorise the results in Table 11-1 into two groups. The first group includes relatively small instances with no more than 100 variables. The second group includes larger instances with more than 100 variables. In the first group of problems all heuristics have a perfect success rate of 100%. The IncHH performance is close to the performance to other heuristics regarding the number of flips used. However, IncHH was able to significantly outperform the

other heuristics solving more instances from the second group as shown also in Figure 11-4.

5. Conclusions

We have used the GP-HH framework for evolving customised SAT heuristics which are used within the Inc* algorithm. GP has been able to evolve high-performance heuristics for SAT problems.

In future work, we will try to generalise the framework to other problem domains, including scheduling, time tabling, TSP, etc. Also, we will test the algorithm on diffident types of SAT benchmarks (e.g., structured and handcrafted SAT problems). We also want to study the effect of grammar design on HH frameworks in more detail.

6. Acknowledgements

The authors acknowledge financial support from EPSRC (grants EP/C523377/1 and EP/C523385/1).

References

Abbass, H. A. (2001). MBO: Marriage in honey bees optimization - A haplometrosis polygynous swarming approach. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 207–214, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press.

Bader-El-Den, Mohamed Bahy and Poli, Riccardo (2007). A GP-based hyper-heuristic framework for evolving 3-SAT heuristics. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1749–1749, London. ACM Press.

Bader-El-Den, Mohamed Bahy and Poli, Riccardo (2008). Inc*: An incremental approach to improving local search heuristics. In *EvoCOP 2008*. Springer. (to appear).

Bader-El-Din, M. B. and Poli, Riccardo (2007). Generating SAT local-search heuristics using a GP hyper-heuristic framework. *Proceedings of the 8th International Conference on Artificial Evolution*, 36(1):141–152.

Boyan, J. and Moore, A. (2000). Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112.

Burke, E. K., Hyde, M. R., and Kendall, G. (2006a). Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *LNCS*, pages 860–869, Reykjavik, Iceland. Springer-Verlag.

Burke, E. K., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S. (2003a). Hyper-heuristics: an emerging direction in modern search technol-

ogy. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer Academic Publishers.

Burke, E. K., Kendall, G., and Soubeiga, E. (2003b). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470.

Burke, E. K., Petrovic, S., and Qu, R. (2006b). Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132.

Cowling, P., Kendall, G., and Han, L. (2002). An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In Fogel, David B., El-Sharkawi, Mohamed A., Yao, Xin, Greenwood, Garry, Iba, Hitoshi, Marrow, Paul, and Shackleton, Mark, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1185–1190. IEEE Press.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

Fukunaga, A. (2002). Automated discovery of composite SAT variable selection heuristics. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 641–648.

Fukunaga, A. (2004). Evolving local search heuristics for SAT using genetic programming. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103, pages 483–494, Seattle, WA, USA. Springer-Verlag.

Gent, I. P. and Walsh, T. (1993). Towards an understanding of hill-climbing procedures for SAT. In *Proc. of AAAI-93*, pages 28–33, Washington, DC.

Gottlieb, J., Marchiori, E., and Rossi, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evol. Comput.*, 10(1):35–50.

Hoos, Holger H. and Stützle, Thomas (2000). Local search algorithms for SAT: An empirical evaluation. *J. Autom. Reason.*, 24(4):421–481.

Marchiori, E. and Rossi, C. (1999). A flipping genetic algorithm for hard 3-SAT problems. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 393–400, Orlando, Florida, USA. Morgan Kaufmann.

Poli, Riccardo, Woodward, John, and Burke, Edmund (2007). A histogram-matching approach to the evolution of bin-packing strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation*, Singapore. accepted.

Rossi, C., Marchiori, E., and Kok, J. N. (2000). An adaptive evolutionary algorithm for the satisfiability problem. In *SAC (1)*, pages 463–469.

Selman, B. and Kautz, H. (1993). Domain-independent extensions to GSAT: solving large structured satisfiability problems. In *Proceedings of theInternational Joint Conference on Artificial Intelligence(IJCAI-93)*, ChambÃ©ry, France.

Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, Seattle.

Selman, B., Levesque, H. J., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In Rosenbloom, Paul and Szolovits, Peter, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California. AAAI Press.

Silva, D. L., O'Brien, R., and Soubeiga, E. (2005). An ant algorithm hyper-heuristic for the project presentation scheduling problem. In Fogel, David B., El-Sharkawi, Mohamed A., Yao, Xin, Greenwood, Garry, Iba, Hitoshi, Marrow, Paul, and Shackleton, Mark, editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, pages 92–99.

Chapter 12

CONSTRAINED GENETIC PROGRAMMING TO MINIMIZE OVERFITTING IN STOCK SELECTION

Minkyu Kim[1], Ying L. Becker[2], Peng Fei[2] and Una-May O'Reilly [3]

[1]*Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139;* [2]*Advanced Research Center, State Street Global Advisors, Boston, MA 02111;* [3]*Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.*

Abstract Genetic programming has been shown to be an effective technique for developing quantitative models to rank stocks with respect to their future returns. Stock selection models derived via genetic programming can often outperform models based on more traditional linear factor combination methods. However, genetic programming also poses an inherent challenge when applied to financial markets: its powerful optimization capability may result in very complex models fitted to historical data that do not remain stable as future data is realized. One approach in previous research has been to address the issue of overfitting by carefully selecting the algorithm's parameters and splitting the test data into different subgroups for model selection and validation purposes. In this study we present a constrained genetic programming framework for stock selection model development that significantly improves on current approaches to minimize overfitting. With careful design of genotype structure and genetic operators, the framework promotes effective stochastic searches that are tightly constrained within a desired function space of limited complexity. Evaluation of resulting stock selection models shows that the proposed framework produces composite factors, i.e., building blocks that reveal some unexpected financial insights. More importantly, the constrained genetic programming reduces data overfitting, yielding models that are more robust across the held-out data set than standard genetic programming models.

Keywords: equity market, stock selection, quantitative asset management, data overfitting.

1. Introduction

The perspective of genetic programming (GP) practitioners on the competence of GP can often be different from that of GP theoreticians. What theo-

reticians seek in a competent GP is the attributes that rely less on some specific domain knowledge and thus can be well applied to a variety of problems with different features. Practitioners, however, start with a specific problem to solve, whose characteristics are usually known to some extent. Hence, what matters more to GP practitioners is how effectively those known properties of the problem can be exploited in the GP framework, leading to a superior performance just for the problem at hand. Application of GP in quantitative financial investment is no exception.

There has been an increasing amount of interest in applying GP, or evolutionary algorithms more broadly, to various investment tasks, for instance, analysis of foreign exchange market (Neely et al., 1997; Lawrenz and Westerhoff, 2003), trading rules in stock market (Allen and Kajalainen, 1999; Li and Tsang, 1999), stock price forecast (Kaboudan, 2000), among many others. In previous work, GP's success has depended upon finding the appropriate problem setup: what GP functions and terminals to employ, particular selection of the training and test sets, cross validation data, fitness measures, etc. Or, what GP parameter settings work most effectively: population size, crossover rate, etc. Researchers at State Street Global Advisors have studied the application of GP for development of effective stock selection models (Zhou, 2003; Caplan and Becker, 2004; Becker et al., 2006; Becker et al., 2007). Along the same direction, in this study we further investigate the challenging issues in stock selection and seek to address them by designing GP mechanisms specifically for the problem.

Quantitative investment strategies employ mathematical models to make investment decisions such as which stocks to buy or sell for their portfolios. The models tend to predict the future performance of the interested securities based upon a number of financial and statistical metrics. The main challenge here is to construct a stock selection model that captures the most effective factors among vast amounts of available information and combines them in an optimal manner to produce reliable and robust forecasts of the securities' future returns in unknown market situations. What matters most here is how to find a particular combination of statistical and financial factors that reflects certain features of financial markets with economic and investor behavioral intuitions, rather than optimally fitting the historical data. Such overfitted models are very likely to fail in predicting future performances often with drastic economic scales. Therefore, the issue of overfitting, which is already a significant subject in statistics and machine learning in general, should be dealt with much more seriously than in other typical symbolic regression problems.

In this regard, stock selection poses an inherent challenge to GP because GP's search mechanism exploits solutions that are judged to be high performance on the given historical data. Combined with the strong tendency to bloat, and a lack of fine-grained control of solution size and structure, GP can evolve very

complicated models without meaningful economic sense that lack the capability to forecast future returns. However, a GP mechanism specifically designed to avoid such overfitting did not exist. Instead, most existing GP approaches rely on an ad hoc setup procedure that divides the data set into subgroups used for model selection and validation purposes and the standard bloat control methods, such as the one by Koza (Koza, 1992), to bound the complexity of the solution.

In this study we propose a GP framework with constrained genotype structure and genetic operators to address the issue of data overfitting specifically for the stock selection model development.

The rest of this paper is organized as follows. Section 2 describes the problem formulation with the discussion of the issue of overfitting in financial modeling and practical methods used by industrial practitioners. Section 3 proposes the constrained GP framework in terms of genotype structure and genetic operators. Section 4 describes the details of our research methodology. In Section 5, we present the resulting models and discuss each model's quality and its performance. Section 6 concludes the paper with the discussion of future work.

2. Problem Formulation

The core part of the quantitative stock selection process is to assign to individual securities a numerical score that ranks its attractiveness in terms of future performance relative to its peers in the given investment universe. The mathematical function used to calculate the score, which we refer to as a *stock selection model*, involves a number of metrics such as underlying firms' intrinsic valuation, operational and management efficiencies, growth rate persistency and earnings estimates, as well as the securities' current and past market performances.

A common feature shared by most traditional stock selection models, for instance, CAPM or Fama-French three-factor model, is that the selected factors are linearly combined. Such linearity, though seemingly simplistic, is convenient and intuitive, so a linear specification has been broadly accepted as a default assumption used in quantitative model development. However, such linear models may not fully express the factors' interactions or power and thus may fail to reflect the complexity of the true relationships.

We aim to apply GP to development of novel stock selection models that are not restricted within the linear framework. In doing so, a special care must be taken because the higher model complexity introduced by exploring beyond linearity may also increase the likelihood of yielding the overfitted models. Therefore, the challenge is how to best describe the nature of the capital market by balancing complexity and parsimony within the GP framework. Before we proceed to introducing our constrained GP framework, let us further investigate the issue of overfitting in finance.

The Issue of Overfitting in Finance

In statistics, overfitting means that a statistical model is fitted perfectly only because it expresses excessive complexity or uses too many dimensions relative to the dimensionality of the available data. Although the relationships described by a model with overfitting appear statistically significant, they are actually spurious and irrelevant. In terms of forecasting, a fitted model using historical data can do a good job of explaining the past, but will do poorly to predict the future. As pointed out in (Hooker, 2006), many standard quantitative techniques can bias naïve users toward overfitted models. Thus, cautions have to be taken in the model development process by controlling a model's degrees of freedom, how much the model's structure is allowed to conform to the data's structure, and the magnitude of model error compared to the expected level of noise in the data.

Quantitative investment heavily relies on a large number of observations for statistical tests. In the past decades, the rapid growth of computer technology has provided investment professionals with tremendous capability of processing and analyzing large scale financial data. The investment researchers strive hard to find statistically significant correlations and patterns between financial data and future stock price movement. In finance, only the stock return predictors discovered using historical data and confirmed over different live validation periods are considered stock market anomalies. Unfortunately, it seems not easy to prove that the historical relations will repeat themselves in the future due to a number of reasons. First of all, some relations discovered from a given sample appear to exist by a random chance so that they are unlikely to persist. Secondly, various business cycles and regime/style shifts can easily drive current market conditions away from the near past. For example, business growth can rotate within a cycle of contraction, stagnation and acceleration; a regime can encompass value/growth, small/large, up/down, or high/low volatility market environments. Thirdly, some of the unique events, regulations and crises that have impacted market before are unlikely to occur again in the future. For example, the recent U.S. Securities and Exchange Commission Regulation Fair Disclosure (Reg. FD) policy has changed the way corporations report their fundamentals because of rule changes in accounting and governance. Consequently, there are potential problems in making a leap from a back-tested strategy to successfully investing in future real market conditions. Since the uniqueness of stock market anomalies may rely on the sample provided, overfitting can well satisfy backtesting in the past, but likely disappoint validation in the future. For the asset management business, the biased opinion caused by overfitting can fool the stock selection process and put large assets at risk.

In all respects, care must be taken to guard against data overfiting in any quantitative stock selection model development process. One effective way

of avoiding the biases caused by overfitting is splitting the historical data to in-sample and out-of-sample portions. In-sample data is used to discover relationships, and out-of-sample data is used to reexamine them. How to partition the backtested samples is very critical. If the time span of the in-sampled data is too long, the information included may be no longer relevant. If too short, a sample can be dominated by a particular regime. Thus, sufficient length of data is required. What emerges is the careful, thoughtfully guided development of quantitative models that assures trust. We think GP aids as much with yielding investment intuitions, "building blocks", that can be verified by an expert financial analyst, as it does with deriving one entire useable model. So, overfitting *and* comprehensibility are equally vitally important. Any result that seems to reveal a stock market anomaly but does not bear an appropriate economic interpretation must be rejected.

3. Constrained Genetic Programming

Comprehensibility plays a crucial role in distinguishing intuitive investment strategies reflecting certain features of the capital market from those just fitting the historical data tightly. Thus, we investigate how to maintain comprehensibility of the stock selection models throughout the iteration of GP, particularly focusing on the following two aspects.

First, stock selection models must have limited complexity to reflect certain intelligible investment intuitions. The predominantly popular mechanism to control the model complexity is the one introduced by Koza that restricts the breeding to only produce children less than some maximal tree depth (Koza, 1992). Given that the individual factors involved in our stock selection model development are already sophisticated investment signals, the desired number of factors in a model with acceptable complexity rarely exceeds 10, and should definitely be below 20. In binary trees, this range translates to depth of 4 or 5, which is much less than the typical maximal depth of 17 in the Koza-style depth limiting method.[1] With such a low depth limit, crossover may become inefficient and cause some useful factor combinations to be lost, as will be discussed more in depth below.

In addition, ensuring comprehensibility may require something more than just limiting the number of factors. For instance, one may have an investment idea that some type of factors are likely to have nonlinear interactions among themselves but not with those of different types. Or one may like to divide the factors into a number of groups and ensure that every group is appropriately represented by some factors in the final stock selection model. However, it

[1]Note that this still defines a search space of significant size because of the very large number of possible factors.

is hard to utilize such intuitions within the standard GP framework where the crossover points are randomly chosen.

We thus propose a tightly constrained GP in terms of the genotype structure and genetic operators specifically designed for the stock selection problem.

Genotype Structure

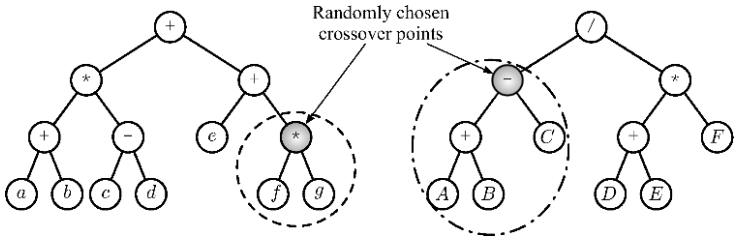
To represent the stock selection models, as in the conventional GP, we use binary[2] trees with the interior nodes having operators and the leaf nodes financial factors. However, we put a very strict constraint on the complexity of the genotypes by never allowing the trees to grow. As mentioned above, we have a very clear criterion as to the complexity of the desired final stock selection models. Therefore, for the initial population we generate full binary trees with the fixed depth, typically to either 4 or 5. While keeping the depth of the trees fixed, we may still allow some flexibility in the number of factors in stock selection models by introducing the *null* factor that serves as the identity element for any operator; it is treated as zero if the associated operator is summation or subtraction, and one if multiplication and division. In order to maintain the fixed-depth trees throughout evolution, we also need to adjust the genetic operators accordingly as we will discuss next.

Genetic Operators

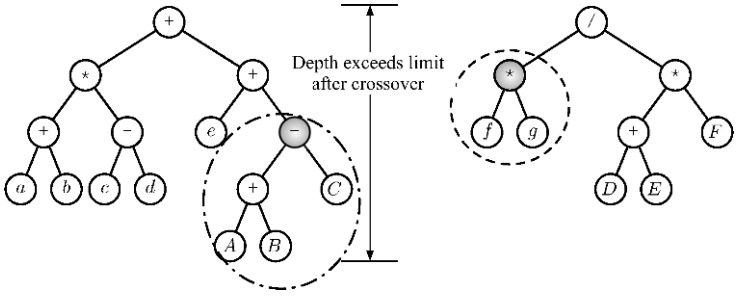
Let us first consider the standard GP crossover operator, where a crossover point is randomly selected in each of the parent trees and the subtrees below the crossover points are swapped. Depending on the location of the chosen crossover points, the depth of the resulting child trees may vary widely. When combined with the standard depth restriction method that rejects children whose depth exceeds the given limit, the breeding process may become very inefficient after some initial generations because one of the children in any mating pair will be almost always rejected (see Figure 12-1). Moreover, since each rejected child tree is replaced with one of its parents, either the subtree it inherited from a parent tree or the remaining part of the other parent tree above the crossover point will be lost. These effects become more pronounced especially when the depth limit is quite small as in our case.

In our new crossover operator, while traversing the trees in depth-first order, we randomly exchange the subtree rooted at each visited node with the subtree located exactly at the same position in the mating counterpart (see Figure 12-2). The probability of subtree exchange depends on the depth of the node. Specifically, we assign smaller probabilities to the nodes closer to the root to preserve the broad structure of the stock selection models; the probability of

[2]Only binary operators are allowed in this study.



(a) Crossover points are randomly chosen separately in each tree of the mating pair.



(b) Left tree is rejected after crossover and replaced with one of the parent trees.

Figure 12-1. Standard crossover with standard depth limiting method with limit 4.

subtree exchange is 1/2 at the leaf nodes and is halved for each level up. In a sense, it is similar to the uniform crossover operator used in genetic algorithms except that the subtrees rather than just the nodes are exchanged. For mutation, we traverse each tree that went through crossover and randomly reassign the contents of the subtree rooted at each node with the given fixed probability.

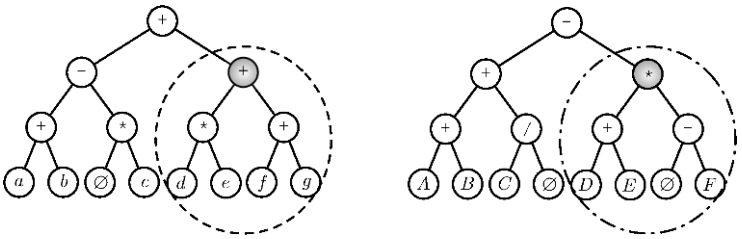


Figure 12-2. Constrained crossover swaps subtrees at the same location.

Note that the key characteristic of the constrained GP is that the genotype structure, i.e., the tree size and nodes' relative locations in the tree, is preserved

throughout the whole evolution process. With this constrained GP, the ideas discussed above can now be accommodated as follows. Suppose that the factors are divided into a number of groups. Then, we can allow only the factors from a particular group in a specific subtree. If one likes to avoid nonlinear interactions among factors across different groups, one can simply restrict the top operators above the subtrees to the linear operators.

4. Research Methodology

Financial Data

In this study, we use the stocks that are components of the S&P 500 index as our investment universe, excluding the financials and utilities sectors, which have very unique business models. The monthly financial data and stock return time series used for the research are downloaded from the Compustat database for the period January 1990 to December 2006. In order to construct the potentially effective stock selection factors, we chose 65 financial variables from a variety of classes, such as fundamentals from balance sheets, income statements, earnings estimates, price returns and market perspectives on a monthly basis. For each given date, there are around 350 investable stocks, each of which is described by the 65 variables.

We use a similar structure as in (Neely et al., 1997) where the iteration is divided into three steps: training, selection and validation (see Figure 12-3). The financial time series data are split into three sets to be used in those three steps as follows. The training step takes the half of the data samples chosen randomly from January 1990 to October 2001, and the remaining half for the selection step. The data from January 2002 to December 2006 are held out for the validation step. Note that, given that one-month forward return is used to measure the performance, the full two months, November and December 2001, are intentionally left out to allow the completely separated out-of-sample test in the validation step.

Objective Function

For a given stock selection model in the population, we calculate the score of the stocks at each time period in the given data set. At each month, we construct a market-neutral portfolio that is long on the top 10% of stocks and short on the bottom 10% based on the calculated scores. We assume that we hold the portfolio until rebalancing it the same way at the next month. Hence, each portfolio produces a return over a one-month horizon.

Our objective is to maximize the annualized average return of these portfolios divided by the annualized standard deviation, which is referred to as the information ratio (IR) expressed as below:

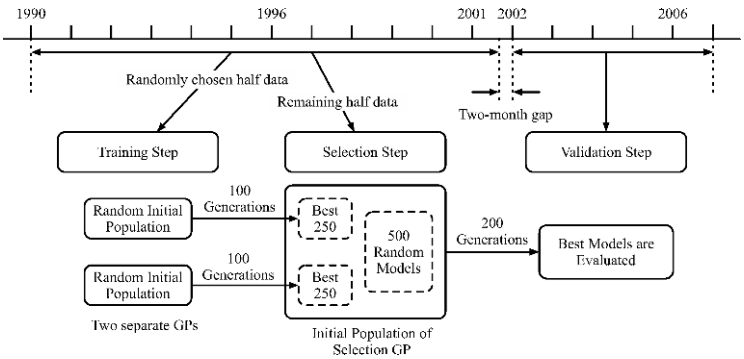


Figure 12-3. A single run consists of three steps using different data sets.

$$\text{spread}(t) = \frac{\sum_{i=1}^{\lfloor 0.1N \rfloor} R_i(t) - \sum_{i=\lceil 0.9N \rceil}^{N} R_i(t)}{\lfloor 0.1N \rfloor}, \tag{12.1}$$

$$\text{IR} = \frac{(\overline{\text{spread}})_{\text{annual}}}{(\sigma_{\text{spread}})_{\text{annual}}}, \tag{12.2}$$

where N is the total number of stocks in the universe and $R_i(t)$ is the one-month forward return on the i-th ranked stock at time t. We use the calculated IR as the fitness value of each stock selection model in the population.

Implementation Details

For better comprehensibility, we choose to use only four basic arithmetic operators: addition, subtraction, multiplication and division. At the terminal nodes, we do not allow numerical coefficients, whose most effective use in GP is still not clear. Instead, we preprocess the data so that the ranges of the factors match one another, and if necessary, we may introduce and adjust the coefficients later in a separate procedure.

As illustrated in Figure 12-3, a single run of the simulation consists of the following procedures: For the training step, we run the constrained GP twice with a randomly generated population of size 1000 for 100 generations. From each resulting population we take the top 250 models and add another 500 randomly generated models to make the initial population for the selection step. The constrained GP is run for 200 generations with the selection data, after which a number of best-performing stock selection models are transferred to the validation step. We perform the out-of-sample test with the selected stock selection models based on the validation data. In all GP runs, the tree depth

is set to 4 and tournament size to 2. Crossover and mutation rates are 0.9 and 0.02, respectively.

For comparison, we also perform the same simulations with the standard GP where the following mechanisms are used: the ramped half-and-half tree generation method (Koza, 1992), the standard depth limiting scheme (Koza, 1992) with limit 4, and the conventional genetic operators (Koza, 1992) with the typical parameter values.

5. Results and Discussions

There have been numerous finance studies on predicting the cross-sectional stock returns based on understanding of the capital market structures, investors' behaviors, firms' fundamentals and growth, as well as analyst opinions of the firms' future earnings. For quantitative model development, not only is it important to recognize and select individual factors that have stock selection capability, but also to characterize their interactions and joint power. In this study, we let the GP framework select and compose stock selection factor models based upon the 65 individual variables, each of which has some degree of selection power.

For the proposed constrained GP and the standard GP framework, we performed three independent runs, each going through training, selection and validation steps. We then picked the run in which the best-of-run stock selection model has highest information ratio (IR) at the validation step.

Resulting Models

For the constrained GP, the best stock selection model, which we refer to as Model C, has the following format:

$$\text{Model } C = V_1 \cdot V_2 - L_1 \cdot L_2 + G_1 + G_2.$$

The best model selected by the standard GP, denoted by Model S, has more complex terms involved in nonlinear operations as follows:

$$\text{Model } S = \frac{A_1 \cdot V_1}{A_2 \cdot L_2} + \frac{V_1}{L_3}.$$

The factors selected by the above two models are summarized in Table 12-1, while their details are intentionally omitted for proprietary reasons.

These factors can be categorized into four groups: valuation, growth, leverage, and analyst. Valuation factors compare a firm's fundamental value to its market price. An undervalued stock, or "cheap" stock, presents a good investment opportunity. Both Models C and S include valuation factors that have well defined economic intuition by themselves. Operating cash flow yield (V_1) uses the traditional rules of fundamental analysis of a firm's financial statement. Cash flows are good measures of how much money a firm is producing

Table 12-1. Summary of factors appearing in resulting stock selection models.

Factor	Description
Valuation	
V_1	Operating cash flow yield
V_2	Dividend yield
Growth	
G_1	Operating profit margin
G_2	Earnings growth margin
Leverage	
L_1	Debt ratio
L_2	Net debt ratio change
L_3	Net debt ratio
Analyst	
A_1	Analyst disagreement
A_2	Long-term estimate revision

at a sustainable level. The investors would expect that the higher operational cash flow, the more promising of firm delivering high future earnings and eventually high stock returns. Thus, this variable is a proxy of measuring a firm's true strength of generating future earnings regardless of sales. In contrast with Model S, Model C also selects another valuation factor, dividend yield (V_2), and combines it with V_1 to further measure the firm's capability of generating cash and rewarding its share holders. This composite non-linear term makes the valuation factor balanced between two variables. Also, its attribution to the model's forecast power can be estimated separately from the other groups of factors, which is always an appealing aspect for a portfolio manager to explain the selection process rationale to the clients.

Growth variables also have stock return predictabilities. These factors measure a firm's business growth through its improvement of profitability and earnings margin. A stock of a high growth firm is likely to have an upward price movement in the near future. The two growth variables that Model C selects have good investment rationales. High operating profit margin (G_1) and earnings growth margin (G_2) reflect the likelihood of high profit improvement relative to the revenue and a growing earnings generation, respectively. These economic performances will, for sure, be transferred to the firm's stock performance in the future. It is noted that the two variables in Model C are combined together linearly to form a composite growth factor. Again, like the valuation factors, this linear growth factor attribution to the model power can be esti-

mated separately from the other categories of factors and therefore is easy to comprehend.

Another group of variables selected by the above models are the measures of firms' financial leverage. Leverage factors reflect a firm's financial strength as well as distress and its operational efficiency hence a firm's economic performance. The stock returns are associated with this economic performance. Three leverage variables, debt ratio (L_1), net debt ratio change (L_2) and net debt ratio (L_3), are all well related to the firm economic performance and thus to their stock future returns. There are many reasons that stocks of firms with high leverage of debt or increasing debt levels earn low future returns. This negative correlation might indicate that the firm would need more cash than what they had anticipated or planned, or that the underlying operations of the firm are not performing as well as expected, therefore, the firm needs to remedy the situation by borrowing more. Issuing more debt is often preferred over issuing equity for the relative low cost and easy process, such as using existing line of credit. While both models recognize the leverage factors' stock selection power, Model C not only emphasizes the importance of the financial leverage by multiplying two variables, but also attributes the impacts to both the debt level and the change of debt level. This seemingly simple concept reveals innovative and valuable investment insights and moreover, can be easily estimated and explained separately from the other group of factors. It is emblematic of the "building blocks" our constrained GP is capable of discovering.

Many academic and empirical researchers have shown that the stock future performance can be predicted using analyst-related factors such as analysts' forecast of the firm's future earnings and the dispersion of analysts' opinions as to earnings. Investors at large have a certain behavioral tendency that they either react slowly to the new information or reject the information that does not agree with pre-existing beliefs. These underreactions in particular to past earnings news would generate undervalued and overvalued stocks to buy and to sell, respectively. Model S combines two analyst variables, analyst disagreement (A_1) and long-term estimate revision (A_2), nonlinearly with valuation and leverage factors, which certainly increases the complexity of the model and creates difficulty for explanation.

To summarize, while both models selected the variables that have certain investment intuitions, Model C combined them in a more intuitive way such that the contribution of each type of factors to the final model score can be more clearly seen, leading to better comprehensibility. Let us now verify if such an advantage in comprehensibility indeed pays off in terms of reducing overfitting and consequently predicting future returns of the stocks.

Model Evaluation

Intuitively, the degree of overfitting can be measured by the performance deterioration of the stock selection model when applied to the future market prediction. Becker et al. (Becker et al., 2007) proposed erosion as a quantitative measure of such performance deterioration, which is defined as the percentage decrease in the model's IR from the GP's selection step to the validation step. Thus, the higher the model's erosion, the higher the degree of overfitting. In Table 12-2, we report the mean and standard deviation of IRs, as well as the mean erosion, of the 10 best models obtained by the two GP frameworks: i.e., for the constrained (standard) GP, Model C (S) and the next 9 models in terms of IR at the validation step. Whereas the models by the standard GP have a higher average IR in the selection step, it is the constrained GP-based models that have a higher average IR in the validation step. Also, the models by the constrained GP display less scattered performance in both steps than those by the standard GP. Note that the constrained GP consequently leads to almost half mean erosion than the standard GP. This result indicates that it is more likely to have an overfitted model using the standard GP than the constrained GP.

Table 12-2. Summary of IRs on 10 best stock selection models obtained by the constrained and standard GPs.

	Selection		Validation		Erosion
	Mean	Std Dev	Mean	Std Dev	Mean
Constrained GP	2.7484	0.0291	1.9572	0.0178	28.78%
Standard GP	3.2950	0.1826	1.5099	0.0962	53.98%

To further evaluate the two GPs, we analyze the performance of Models C and S with a fresh set of data that has not been used in the GP iteration. Due to the limitation of available years in the post-GP testing period, we use the four-year observations from 1986 to 1989 for this out-of-GP-out-of-sample evaluation. Table 12-3 shows monthly long-short portfolio returns for various lengths of holding periods. A portfolio's hit rate is the percentage of time periods when the long-short portfolio has a positive return. In terms of average monthly portfolio returns and IRs for different holding periods, Model C consistently outperforms Model S. Except for very short-term (1-month) holding period, Model C has higher hit rate than Model S for either 3-month or 6-month holding horizon. Overall, Model C generated by the constrained GP demonstrates better stock selection capability than Model S by the standard GP in the testing period.

These results suggest that the better comprehensibility of the stock selection models generated by the constrained GP may indeed lead to better forecasting capability. While we can conclude that the constrained GP framework has

Table 12-3. Total portfolio returns from buying top decile and short selling bottom decile stocks in the period of 1986-1989.

Decile Spread Returns		Model C	Model S
1-month	Mean	0.96%	0.73%
	Std	2.73%	2.08%
	IR	1.22	1.22
	Hit Rate	60%	69%
3-month	Mean	2.53%	1.53%
	Std	4.78%	3.60%
	IR	1.06	0.85
	Hit Rate	79%	67%
6-month	Mean	4.27%	2.08%
	Std	7.72%	5.14%
	IR	0.78	0.57
	Hit Rate	69%	60%

demonstrated a certain degree of data overfitting reduction in the stock selection model development, more effort is still needed to make a forecast model performance sustainable in predicting future realities. The nature of stock market determines that different time period may be concentrated on different business cycles and market regimes. Further work can be focused on either improving sampling method or developing better multi-objective functions to characterize the dynamics of market conditions.

6. Conclusions and Future Work

A continuing challenge in developing stock selection models using GP process is minimizing overfitting. Earlier research has addressed the issue by carefully selecting the algorithm's parameters and splitting the test data into different sub-groups for model selection and validation purposes. A model that is perfectly overfitted in GP evolution process often performs poorly at validation step, let alone in live performance. In this study, we present a constrained genetic programming framework for developing stock selection models that features significant advances from the legacy approach to minimize overfitting issues. This approach aims to ensure the comprehensibility and reflect the investment insights throughout the evolution process. With careful design of genotype structure and genetic operators, the framework promotes effective stochastic search tightly constrained within the desired function space of limited complexity. The proposed constrained GP framework finds financially

significant building blocks that reveal unobvious financial insights. It reduces data overfitting and generates more robust models that generalize better to the held-out test data set than a standard GP framework.

Although the improvement of our constrained GP framework takes a step forward on minimizing data overfitting, the challenge of further reducing performance deterioration remains. Because of the unique features of the stock market environment, the GP evolutionary process must also cope with the dynamics of market characteristics and the resulting impact on a stock's price movement. We intend to concentrate our future work on studying better multiobjective functions that allow us to achieve tradeoffs among models. This will allow us to select a model according to a different business cycle or market regime.

Acknowledgment

We appreciate Mark Hooker, Senior Managing Director of the Advanced Research Center at State Street Global Advisors, for his continuous support.

References

Allen, F. and Kajalainen, R. (1999). Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, 51(2):245–271.

Becker, Ying, Fei, Peng, and Lester, Anna M. (2006). Stock selection : An innovative application of genetic programming methodology. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12, pages 315–334. Springer, Ann Arbor.

Becker, Ying L., Fox, Harold, and Fei, Peng (2007). An empirical study of multiobjective algorithms for stock ranking. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 14, pages 241–262. Springer, Ann Arbor.

Caplan, Michael and Becker, Ying (2004). Lessons learned using genetic programming in a stock picking context. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 6, pages 87–102. Springer, Ann Arbor.

Hooker, Mark (2006). Complexity and parsimony in alpha modeling. State Street Global Advisors Research Library (available at `http://www.ssga.com/library/resh/markhookercomplexityandparsimony20060221/page.html`).

Kaboudan, M. A. (2000). Genetic programming prediction of stock prices. *Computational Economics*, 6(3):207–236.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Lawrenz, C. and Westerhoff, F. (2003). Modeling exchange rate behaviorwith a genetic algorithm. *Comput. Econ.*, 21(3):209–229.

Li, Jin and Tsang, Edward P. K. (1999). Investment decision making using FGP: A case study. In Angeline, Peter J., Michalewicz, Zbyszek, Schoenauer, Marc, Yao, Xin, and Zalzala, Ali, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1253–1259, Mayflower Hotel, Washington D.C., USA. IEEE Press.

Neely, Christopher J., Weller, Paul A., and Dittmar, Rob (1997). Is technical analysis in the foreign exchange market profitable? A genetic programming approach. *The Journal of Financial and Quantitative Analysis*, 32(4):405–426.

Zhou, Anjun (2003). Enhance emerging market stock selection. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practise*, chapter 18, pages 291–302. Kluwer.

Chapter 13

CO-EVOLVING TRADING STRATEGIES TO ANALYZE BOUNDED RATIONALITY IN DOUBLE AUCTION MARKETS

Shu-Heng Chen[1], Ren-Jie Zeng[1] and Tina Yu[2]

[1]*National Chengchi University, Taipei, Taiwan 116.*
[2]*Memorial University of Newfoundland, St. John's, NL A1B 3X5 Canada.*

Abstract We investigate double-auction (DA) market behavior under traders with different degrees of *rationality* (*intelligence* or *cognitive ability*). The rationality of decision making is implemented using genetic programming (GP), where each trader evolves a population of strategies to conduct an auction. By assigning the GP traders different population sizes to differentiate their cognitive ability, through a series of simulations, we find that increasing the traders' intelligence improves the market's efficiency. However, increasing the number of intelligent traders in the market leads to a decline in the market's efficiency. By analyzing the individual GP traders' strategies and their co-evolution dynamics, we provide explanations to these emerging market phenomena. While auction markets are gaining popularity on the Internet, the insights can help market designers devise robust and efficient auction e-markets.

Keywords: bounded rationality, zero-intelligence, agent-based modeling, human subject experiments, auction markets design, double-auction markets, macroeconomics, trading strategies, software agents, market simulation, market efficiency.

1. Introduction

While genetic programming (GP) has been applied to solve various practical problems in the pursuit of financial rewards, there are other types of applications where GP has made important non-financial contributions. This chapter presents one such kind of application where GP has been used to co-evolve trading strategies in an artificial double-auction (DA) market. The objective of the study is to understand the DA market behavior under traders with various degrees of *rationality* (*intelligence* or *cognitive ability*) as well as the learning

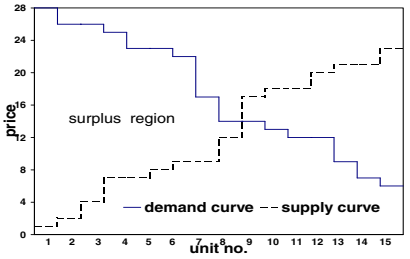


Figure 13-1. Demand and supply curves.

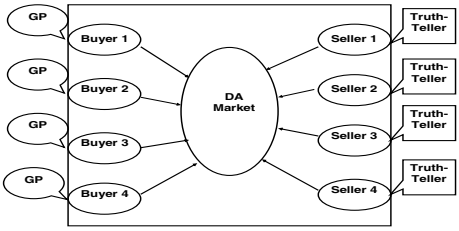


Figure 13-2. The 8 traders' DA market.

behavior of individual traders. Such insights are important to the design of robust and efficient real-world DA markets, where the maximum possible profit is realized. As the number of Internet auction markets (e.g., eBay and Amazon) increases and these markets become more diverse, this GP application can enhance the design of future auction e-markets.

The remainder of this chapter is organized as follows. Section 2 explains DA markets and Section 3 discusses the motivations for this work. In Section 4, our approach to using GP population size to implement traders' rationality is presented. Section 5 details the setups needed to carry out the experiments. The DA aggregate market behavior results are then discussed in Section 6. We analyze the trading strategies that GP evolved in Section 7. Finally, Section 8 concludes the chapter.

2. Market Efficiency and Double Auction Markets

In a standard market environment, the demand for and supply of a commodity can be satisfied under different market prices. When the price is high, the demand decreases and when the price is low, the supply decreases. Adam Smith's *invisible hand* will determine the market price such that both demand and supply are satisfied.

Figure 13-1 gives the demand and supply curves of a standard market. The demand curve gives the maximum price that consumers can accept for buying a given commodity (e.g. cotton, electricity), and the supply curve gives the minimum price at which the producers are willing to sell that commodity. For example, in Figure 13-1, the maximum price that buyers can pay for the second unit is 26, and the minimum price that sellers would accept is 2.

Transactions only take place when the market price is below the demand curve and above the supply curve. The area between the demand and supply curves is the surplus region generated by the transactions. For example, for the first unit, the highest acceptable price for the buyers is 28, and the lowest acceptable price for the sellers is 1. So, the difference between the two (surplus)

is $28 - 1 = 27$. Similarly, for the second unit, the highest acceptable price for the buyers is 26, and the lowest acceptable price for the sellers is 2; hence the surplus is $26-2 = 24$. The distribution of the surplus depends on the transaction price, which must lie between the two acceptable limits. For example, if the transaction price of the first unit is 20, the surplus distributed to the consumer is $28 - 20 = 8$, and the rest, $20 - 1 = 19$, is distributed to the producer.

In an efficient market, a commodity's price is between the demand and supply curves so that all potential surpluses can be fully realized. When the market price is outside the curves, no transaction can occur. Consequently, the commodity stays with the sellers leaving consumers unsatisfied. This market is not efficient. DA is one type of market structure that results in high market efficiency, and hence is very popular in the world. For example, the New York Stock Exchange (NYSE) and the Chicago Mercantile Exchange are organized as DA markets.

In a DA market, both buyers and sellers can submit bids and asks. This is opposed to only buyers shouting bids (as in an *English Auction*) or only sellers shouting asks (as in a *Dutch Auction*). There are several variations of DA markets. One example is the *clearinghouse* DA of the Santa Fe Token Exchange (SFTE) (Rust et al., 1993) on which this work is based.

On the SFTE platform, time is discretized into alternating *bid/ask* (BA) and *buy/sell* (BS) steps. Initially, the DA market opens with a BA step in which all traders are allowed to simultaneously post bids and asks. After the clearinghouse informs the traders of each others' bids and asks, the holders of the *highest bid* and *lowest ask* are matched and enter into a BS step. During the BS step, the two matched traders carry out the transaction using the *mid-point* between the *highest bid* and the *lowest ask* as the transaction price. Once the transaction is cleared, the market enters into a BA stage for the next auction round. The DA market operations are a series of alternating BA and BS steps.

3. Motivation and Related Work

Economic markets are complex systems where the market behavior is the result of the aggregate of many traders' behavior. Due to the interdependence of the traders' actions, it is not easy to isolate or analyze any individual trader's auction strategies based on the market's overall performance. However, understand these trading strategies can help identify and isolate the causes of market inefficiency. Recently, simulation approaches have been developed to study individual traders behavior in economic markets.

When conducting DA simulations for market analysis, human traders may reflect the real market more closely than software agents. However, when the study involves traders' rationality, there is no consensus on how to represent and measure human cognitive ability. One simple approach is using traders' background information, such as education and experiences, to determine their

intelligence. Unfortunately, studies have shown that such information does not reflect human rationality in decision making (Duffy, 2006). By contrast, software agents, such as GP systems, can represent traders with different learning characteristics through different parameter settings. Recently, more and more studies have been using software agents to model and simulate DA markets, through an approach called agent-based modeling (ABM). Both ABM and human subject experiments implement controlled "laboratory" conditions to isolate the sources of aggregated market phenomena. The two approaches complement each other and provide information to advance our knowledge of DA market properties (Duffy, 2006; Chen and Tai, 2003).

The interest in investigating DA market behavior under traders with different degrees of rationality was strongly influenced by the concept of *bounded rationality* introduced by Herbert Simon. Prior to that work, much theoretical work on economics was based on the assumption that individuals are *perfectly rational* in their decision making. This assumption is invalid as humans engage in "rational choice that takes into account the cognitive limitations of the decision-maker, limitations of both knowledge and computational capacity" (Simon, 1997). Due to the bounded rationality, humans make suboptimal yet acceptable decisions by creating partial strategies (simple heuristics) based on imperfect information.

Gode and Sunder (Gode and Sunder, 1993) were among the first to study the impact of bounded rationality on DA market performance. In that work, traders had zero-intelligence in that all bids and offers were randomly generated within their budget constraints (i.e., traders were not permitted to sell below their costs or buy above their values). The DA market with only zero-intelligence traders was able to achieve almost 100% market efficiency. Based on these experimental results, Gode and Sunder argued that the rationality of individual traders accounts for a relatively small fraction of the overall market efficiency.

To investigate the generality of Gode and Sunder's result, ABM researchers have conducted similar experiments using zero-intelligence traders in various types of DA markets. For example, Cliff and Burten (Cliff and Bruten, 1997) studied a DA market with asymmetric supply and demand curves. They found that zero-intelligence traders gave rise to poor market efficiency. They then assigned the traders with the ability to use the closing price in the previous auction round to determine the current bid. Such traders, whom they referred to as zero-intelligence-plus, performed better and improved market efficiency. Thus, individual traders' cognitive ability does impact overall market efficiency.

In this work, we are interested in understanding market behavior under traders with different degrees of intelligence. Would increasing traders' intelligence improve the market efficiency? Does a trader learn better in the presence of another intelligent trader? Is a trader using strategies that exploit the market in an unfair manner? The answers to these questions can help DA market designers

implement rules, such as transaction pricing and the order of bid and offer, so that the market is fair and efficient under traders with different qualities.

4. Methodology

Although the representation of human intelligence is still a debatable issue, we find GP population size to be a reasonable parameter indicating a trader's cognitive ability. In recalling the *learning from experience* analogy of (Arthur, 1993), a trader's strategies are influenced by two factors: the trader's original ideas of how to bid/ask, and the experiences he/she learned during the auction process. In GP learning, the population is the brain that contains the possible strategies to be used for the next bid/ask. It is reasonable to argue that a bigger population size gives the learner a larger reservoir to store and process new strategies. We therefore assign traders with different population sizes to differentiate their cognitive ability when evolving trading strategies.

The trading strategies are represented as rules. We provide three types of information for GP to construct these rules (see Table 13-1):

- Past experiences: terminals 1 – 9 and 16 – 17
- Time information: terminals 10 – 11
- Private information: terminals 12 – 14

In this implementation, only transaction information on the previous day/auction round is provided for GP to compose bidding strategies. In our future work, we will investigate if more profitable strategies will evolve when GP is provided with longer memory.

The three types of information are combined using logical and mathematical operators to decide the bidding prices. Table 13-2 lists these operators.

Each GP trader has a population of strategies to use during an auction. These strategies evolve during GP experimental runs. Each GP generation consists of multiple days and on each day one strategy is randomly selected to carry out a series of auction rounds. The fitness of a strategy is the surplus it accumulated from the transactions at the end of the day when all auction rounds were completed. Since the number of days in each generation is twice the population size, each strategy is likely to be selected at least once and has a fitness value.

The quality of the strategy depends on how well it plays with other traders' strategies during the auction. When multiple self-interest GP traders are in the market, they co-evolve their strategies with the goal being to realize the most surplus. Co-evolution dynamics is not easy to control and analyze. A healthy co-evolution dynamics would develop an "arms race" spiral in which each population spurs the other(s) to advance and the result is continuous learning for all populations. However, this does not always happen. Sometimes, the improvement in one population leads to the deterioration of other populations.

Table 13-1. Terminal Set.

Index	*Terminal*	*Interpretation*
1	PMax	the highest transaction price on the previous day
2	PMin	the lowest transaction price on the previous day
3	PAvg	the average transaction price on the previous day
4	PMaxBid	the highest bidding price on the previous day
5	PMinBid	the lowest bidding price on the previous day
6	PAvgBid	the average bidding price on the previous day
7	PMaxAsk	the highest asking price on the previous day
8	PMinAsk	the lowest asking price on the previous day
9	PAvgAsk	the average asking price on the previous day
10	Time1	the number of auction rounds left for today
11	Time2	the number of auction rounds that have no transaction
12	HTV	the highest token value
13	NTV	the second highest token value
14	LTV	the lowest token value
15	Pass	pass the current auction round
16	CASK	the lowest asking price in the previous auction round
17	CBID	the highest bidding price in the previous auction round
18	Constant	randomly generated constant number

Table 13-2. Function Set.

Function	*Function*	*Function*	*Function*	*Function*
+	-	*	%	min
>	exp	abs	log	max
sin	cos	if-then-else	if-bigger-then-else	

It is also possible that all populations will settle into a mediocre stable state, reaching a local optimum and being unable to move beyond it (Koza, 1992).

In this research, we will investigate the global market outcome produced as a result of the co-evolution of multiple GP traders (see Section 6). In addition, the changes in behavior of an individual GP trader due to his/her co-evolution with another GP trader will be analyzed (see Section 7).

The DA Market Environment

The simulation was conducted using the DA system implemented in (Chen and Tai, 2003). We have customized the system in the following areas to meet our research goals:

Table 13-3. Token Value Table.

buyer 1	buyer 2	buyer 3	buyer 4	seller 1	seller 2	seller 3	seller 4
23	28	23	26	2	8	1	7
22	26	14	25	7	12	4	9
12	17	9	14	17	21	9	20
7	13	6	12	18	23	18	21

- There are 4 buyers and 4 sellers, each of whom is assigned 4 private token values for trading. For buyers, these are the 4 highest prices that they are willing to pay to purchase 4 tokens and, for sellers, they are the 4 lowest prices that they accept to sell these tokens. Table 13-3 lists these values.
- We regard all sellers as being truth-teller, who always gave the assigned true token value during an auction. For buyers, however, we have varied the number of GP buyers from 1 to 4. In this way, we can study the macro market behavior and micro individual behavior changes under the co-evolution of a different number of GP buyers. Figure 13-2 gives the setup of this market design.

Our DA market design is based on the *clearinghouse* DA of the Santa Fe Token Exchange (SFTE). Each DA market simulation is carried out with a fixed number of GP generations (g), where each generation lasts n (n= $2 \times pop\_size$) days. On each day, 4 new tokens are assigned to each of the buyers and sellers. The 8 traders then start the auction rounds to trade the 16 tokens. A buyer would start from the one with the highest price and then moving to the lower priced ones while a seller would start from the one with the lowest price and then move to the higher priced ones. The day ends when either all 16 tokens have been successfully traded or the maximum number of 25 auction rounds is reached. Any un-traded tokens (due to no matching price) will be cleared at the end of each day. The following day will start with a new set of 16 tokens.

During each auction round, if the trader is a truth-teller, the bid/ask price is always the current token value (HTV for buyers and LTV for sellers). If the trader uses GP to evolve strategies, one randomly selected strategy from the population will be used to decide the bidding price. The GP trader may decide to $pass$ the round without giving a bid. The same 8 strategies will play 25 auction rounds, during which GP traders may give different bidding price if the selected strategy uses information from the previous round/day.

Once all 8 traders have presented their prices, the highest bid is compared with the lowest ask. If the bid is higher than the ask, there is a match and the transaction takes place using the average of the bid and ask as the final price. The profits for these 2 strategies (the difference between the transaction and the

given private token prices) are recorded. The fitness of a strategy is the accumulated profit from the transactions of its 4 tokens during the 25 auction rounds.

$$F = \sum_{i=1}^{m} |TokenValue_i - TransactionPrice_i|,$$

where m is the number of tokens traded using the strategy. Since one strategy is randomly selected each day to carry out the auction, after n (n= $2 \times pop\_size$) days, each strategy in the population will most likely be selected at least once and has a fitness value. This fitness value decides how each strategy will be selected and mutated to generate the next generation of new strategies.

When evaluating a GP trading strategy, it is possible that the output price is outside the private price range. That is, the traders may sell below the cost or buy above the value of a token. We might interpret this as a strategic way to win the auction. Since the final price is the average of the winning bid and ask, the trader might still make a profit from the transaction. However, such a risk-taking approach has been shown to make the market unstable and to reduce the market efficiency (Chen and Tai, 2003). We therefore enforce the following rule on the price generated by a GP trading strategy.

if Bid $>$ 2$\times$HTV then Bid=HTV

if Ask $<$ (1/2)$\times$HTV then Ask=HTV

These rules protect the market from becoming too volatile and also allow GP to evolve rules that take on a small amount of risk to generate a profit.

Market Evaluation Criteria

In this work, we analyze the following 8 market characteristics based on the daily transactions results. Let $V_{b,i}$ be buyer b's private value of token i, $V_{s,i}$ be seller s's private value of token i and P_i be token i's transaction price. Also, the potential daily surplus (PDS) is the daily market profit when all traders are truth-tellers who use the assigned token values to conduct the auction. This is the maximum possible daily market profit from all transactions.

1. ***Individual Profit***: The accumulated daily profit of each individual.
   - The profit of buyer b is:
     $\pi_b = \sum(V_{b,i} - P_i)$, **i is the token number b bought.**
   - The profit of seller s is:
     $\pi_s = \sum(P_i - V_{s,i})$, **i is the token number s sold.**
2. ***Consumer Surplus(CS)***: The total daily profit of all 4 buyers.

   $CS = \sum \pi_b$, b=1,2,3,4

Table 13-4. GP Parameters.

Parameter	*Value*	*Parameter*	*Value*
tournament selection size	5	elitism size	1
Initialization method	grow	maximum tree depth	5
population size	10,20,30,40,50	number of days	2×pop_size
crossover rate	1.0	subtree mutation rate	0.005
no. of generation	300	point mutation rate	0.045
no. of runs for each setup	90	no. of buyers that are GP trader	1,2,3,4

3. ***Producer Surplus(PS)***: The total daily profit of all 4 sellers.

   $PS = \sum \pi_s$, s=1,2,3,4

4. ***Total Surplus(TS)***: The daily profit of all transactions.

   TS=CS+PS

5. ***Individual Realized Surplus Ratio (IR)***: The proportion of the potential daily surplus (PDS) that is realized by each individual.

   Buyer b's IR=$\frac{\pi_b}{PDS}$
   Seller s's IR=$\frac{\pi_s}{PDS}$

6. ***Consumer Realized Surplus Ratio (CSR)***: The proportion of the potential daily surplus (PDS) that is realized by all 4 buyers.

   CSR=$\frac{CS}{PDS}$

7. ***Producer Realized Surplus Ratio (PSR)***: The proportion of the potential daily surplus (PDS) that is realized by all 4 sellers.

   PSR=$\frac{PS}{PDS}$

8. ***Total Realized Surplus Ratio(TR)***: The proportion of the potential daily surplus (PDS) that is realized by all transactions.

   TR=$\frac{TS}{PDS}$

5. Experimental Setup

Table 13-4 gives the GP parameter values used to perform simulation runs. With 5 different population sizes and 4 different ways to assign GP buyers, the total number of setups is 20. For each setup, we made 90 runs. The total number of simulation runs made was 1,800.

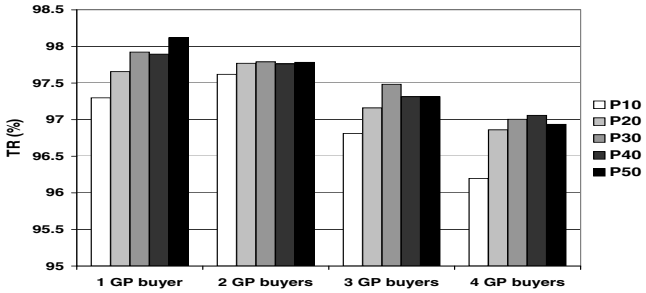


Figure 13-3. Total realized market surplus (TR) under different experimental setups.

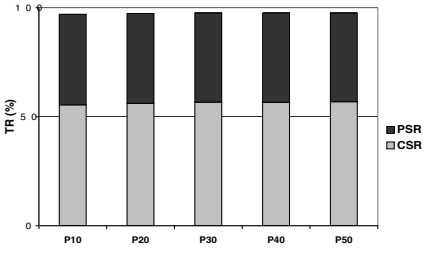


Figure 13-4. CSR vs. PSR under different GP population sizes.

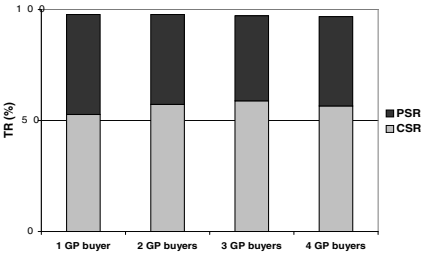


Figure 13-5. CSR vs. PSR under different numbers of GP buyers.

6. Macro Market Behavior Results

The market's total realized surplus ratios (TR) (averaged over 90 runs) under different experimental setups are presented in Figure 13-3. As shown, TR is high (more than 95%) regardless of the size of the GP population and the number of GP buyers in the market. One possible reason is that the market has naive truth-telling sellers, which make the market less competitive. Once the sellers are replaced with more sophisticated traders, the market's efficiency might decline. This will be investigated in our future work.

The results also show that, in general, TR *increases* as the GP population size increases, but *decreases* as the number of GP buyers in the market increases. In other words, all things equal, replacing a GP buyer with another more "intelligent" GP buyer increases the aggregated market efficiency. However, replacing a naive truth-telling buyer with a GP buyer decreases the market efficiency. To understand why this is happening, we calculated the market profit distribution between buyers (CSR) and sellers (PSR) under different GP population sizes (Figure 13-4) and different numbers of GP buyers (Figure 13-5).

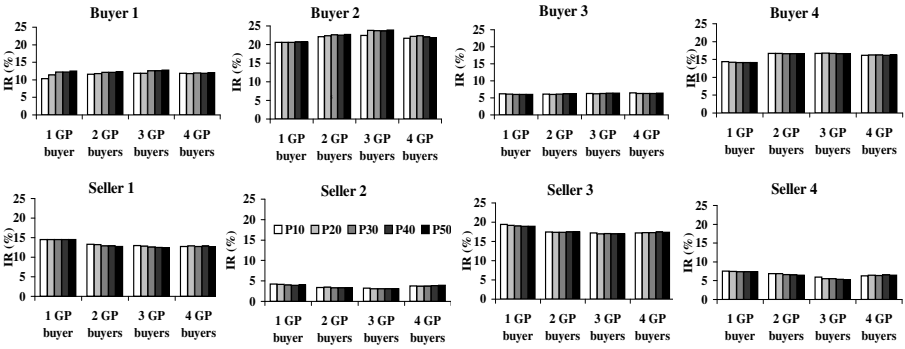


Figure 13-6. Individual realized market surplus (IR) under different experimental setups.

In both figures, buyers allocated more of the total market profit than the truth-telling sellers (CSR>PSR). This indicates that GP produces strategies that are more profitable than the naive truth-telling strategy. Meanwhile, when the number of GP buyers increases, CSR increases and PSR decreases. This indicates that the co-evolution of the GP buyer populations has generated more profitable strategies and strengthened the GP buyers' ability to reduce the sellers' profits. The gain in terms of CSR is, however, less than the loss of PSR. Consequently, the market's overall profits have declined.

By contrast, the naive truth-telling traders have no ability to learn. A "smarter" GP buyer (who has a larger population size) can therefore exploit the simple bidding pattern of these naive traders and increase its own profit. This improved CSR is not completely at the cost of PSR. Consequently, the total market efficiency is increased.

When considering the aggregated market behavior, both the increase in the GP population size and the number of GP buyers empower the buyers to exploit the naive truth-telling sellers. Increasing the GP population size impacts the market at the individual level, as it assigns a higher-degree of intelligence to a single buyer. By contrast, increasing the number of GP buyers impacts the entire buyer group, as all GP buyers co-evolve their trading strategies together. The latter approach therefore has a stronger impact on the market's profit allocation and produces a larger increase in CSP than the first approach.

Figure 13-6 shows the profits of each individual (IR) under different population sizes and numbers of GP buyers. On the buyers' side, either increasing the population size or the numbers of GP buyers has a positive impact on the profit. When a buyer is switched from truth-telling to GP, the buyer's profit increases. Meanwhile, other GP buyers also obtain increased profits. This again

Table 13-5. Distribution of daily profit generated by 18,000 strategies (pop_size 10).

Profit	-0.5	0	8	10.5	12.5	14	14.5	17	18	18.5	21	Total
Count	18	429	346	5,666	72	8	3,829	16	4	335	7,277	18,000

indicates that the co-evolution dynamics benefits all GP buyers. On the sellers' side, however, the effect is exactly the opposite. All truth-telling sellers face declining profits when "smarter" GP buyers or new GP buyers join the market; their profits are taken away by these newcomers.

Buyer 3 and seller 2 have a noticeably lower IR than the other traders. This is partly due to their tight budget constraints. As shown in Table 13-3, buyer 3 can not pay much for his 4 tokens and seller 2 faces high costs of his 4 tokens. This makes it hard for them to compete with other traders during an auction. Even when they win the auction, the profits from the transactions are low.

7. Individual GP Trading Strategies Analysis

We also analyze individual GP buyer strategies using the experimental results from two different settings. The first is when there is only one GP buyer in the market and the second is when there are two GP buyers in the market. In the first case, the focus is on how GP population size influences the learning of strategies. In the second case, besides population size, we also investigate the co-evolution dynamics of two GP buyers and its impact on the learning of strategies. We address these two cases in two sub-sections.

The data used to conduct our analysis are obtained from the runs with population sizes 10 and 50, which give rise to the largest difference in trader's "intelligence". We collected data on all evolved strategies and their daily profit (π) generated during the last 10 generations. We consider these strategies to be more "mature", and hence to better represent the GP buyers' trading patterns. With 90 runs for each setup, the data set is large enough for us to conduct the statistical analysis.

One GP Buyer Case

When the population size is 10, each generation is $2 \times 10 = 20$ days long. On each day, one strategy is picked randomly from the population to conduct the auction. The total number of auction days for the last 10 generations is therefore $20 \times 10 = 200$. Since we made 90 runs for this setup, the number of strategies picked to conduct the auction is $200 \times 90 = 18,000$. Table 13-5 gives the distribution of daily profit (π) generated by these strategies.

Table 13-6. The 3 most used strategies and the associated information (pop_size 10).

Strategy	Profit	Count	Ratio(Count/18,000)
PMinBid	21	7,277	0.4043
PMinBid	10.5	5,666	0.3148
HTV	14.5	3,829	0.2127
Total		16,722	0.9318

As shown in the table, the strategies that generated profits 21, 10.5 and 14.5 are used for a total of 93% of the auction. It is therefore reasonable to assume that they represent the GP buyer's trading strategies. Table 13-6 presents these 3 strategies and their associated information.

The most highly used strategy is PMinBid: the lowest bidding price on the previous day. Since the strategy that provided the lowest bidding on the previous day can be different, the generated bidding price can also be different. Consequently, this strategy can produce different profit. In our case, PMinBid generates a profit of 21 on 7,277 days and a profit of 10.5 on 5,666 days. After careful examination, we find that if the strategy used on the previous day to give the lowest bidding price is HTV (the highest token value) or NTV (the second highest token value), PMinBid will give a profit of 21 for the current day. If the strategy was PMaxAsk (the highest asking price among all transactions on the previous day), PMinBid gives a profit of 10.5 for the current day.

The other highly used strategy is HTV: the highest token value, which is also equal to truth-telling, as we always trade the token with the highest value first. This strategy gives a profit of 14.5, which is between the two possible profits produced by PMinBid.

Although using historical information (PMinBid) can be unstable (sometimes giving a profit of 21 and sometimes of 10.5), the buyer was found to be willing to take the risk and used it on 70% of the auction days, instead of being a safe truth-teller (HTV), who gained a stable and good profit (14.5). This risk taking did pay off: the number of days that PMinBid gave a profit of 21 was 20% higher than the number of days that gave a profit of 10.5. Overall, the strategy gave a higher profit than telling the truth.

The more intelligent buyer (GP with a population size of 50) has developed slightly different trading strategies under an identical setting. In addition to PMinBid and HTV, another class of strategies has also emerged. These strategies are more complex (longer in length) and give a slightly higher profit of 22 (we will call this group of strategies p-22). Table 13-7 lists the 4 most frequently used strategies by the more intelligent GP buyer together with the associated information. Note that the number of days for each generation is $2 \times 50 =$

Table 13-7. The 4 most used strategies and the associated information (pop_size 50).

Strategy	Profit	Count	Ratio(Count/90,000)
p-22	22	22,381	0.2487
PMinBid	10.5	10,976	0.1220
PMinBid	21	29,779	0.3309
HTV	14.5	7,827	0.0870
Total		70,963	0.7885

100. The total number of auction days (also the number of strategies picked to conduct auction) during the last 10 generations for all 90 runs is therefore $100 \times 10 \times 90 = 90,000$.

We examine this new class of strategies and find that they can be reduced to one of the following two forms:

- `Min PMinBid HTV`
- `If_Bigger_Then_Else PMinBid HTV HTV PMinBid`

These two strategies are identical semantically: combine `HTV` and `PMinBid` and choose the one which gives the lower price. The emergence of this class of strategies indicates that the "smarter" GP buyer is able to fine-tune existing strategies to produce more profitable strategies. This is probably due to its larger population size which allows more diversity in the strategies pool. Operators, such as `Min` and `If_Bigger_Then_Else`, have more opportunities to survive even if the strategies that contain them did not give rise to a very high profit. At a later time, GP is able to use them to fine-tune (through crossover and mutation) existing good strategies to become better ones.

The "smarter" GP buyer also used `PMinBid` more wisely: the percentage of its usage that generated a profit of 21 increased (40% for profit 21 and 31% for profit 10.5 when population size is 10 vs. 33% for profit 21 and 12% for profit 10.5 when population size is 50). This is probably due to this strategy utilizing historical auction information. With the emergence of the more profitable p-22 strategies, `PMinBid` is more likely to make better decisions.

The dependence between the `PMinBid` and p-22 strategies is not one-way. The existence of the p-22 strategies also relies on the presence of `PMinBid` and `HTV` (otherwise, the two less profitable strategies would have been replaced by p-22 strategies). This is not difficult to comprehend as p-22 strategies are composed of `PMinBid` and `HTV`. The co-existence of these 3 groups of strategies (p-22, `PMinBid` and `HTV`) is therefore mutually beneficial. The "intelligent" GP buyer has learned that information and has maintained the diversity (since it has a large enough population space) to make profitable auction decisions.

Table 13-8. Distribution of daily profit of the 2 most used strategies for two GP buyers.

	Pop_size 10				Pop_size 50			
Buyer	Profit	Count	Ratio	Total	Profit	Count	Ratio	Total
1	17	13,042	0.7246		17	53,704	0.5967	
	24	392	0.0218	0.7463	24	11,054	0.1228	0.7195
2	35	9,702	0.5390		35	38,358	0.4262	
	37	1,004	0.0558	0.5948	37	17,731	0.1970	0.6232

Two GP Buyers Case

With 2 GP buyers in the market, the learning dynamics becomes more complicated. Table 13-8 gives the profit distributions of the 2 strategies most used by the two GP buyers under population sizes of 10 and 50.

For GP buyer 1, the strategies most used are the ones that result in a profit of 17 (p-17) and for GP buyer 2, the strategies most used are those that result in a profit of 35 (p-35). This pattern is the same under population sizes 10 and 50. Meanwhile, when the population size was increased from 10 to 50, GP buyer 1 increased the usage of the strategies that realized a profit of 24 (p-24) and GP buyer 2 increased the usage of the strategies that produced a profit of 37 (p-37). These two patterns suggest the following correlation:

- GP buyer 1 used p-17 to auction against the p-35 of GP buyer 2
- GP buyer 1 used p-24 to auction against the p-37 of GP buyer 2

While examining the data, we found that many strategies produced the same profit. Meanwhile, GP buyer 1 used p-24 against many different strategies of GP buyer 2, not just p-37. Nevertheless, the p-17 vs. p-35 and p-24 vs. p-37 pairs appeared frequent enough in the auction rounds that we decided to conduct more of an investigation.

For the p-17 vs. p-35 pair, the most frequently used strategies were `HTV` of GP buyer 1 and `PMax` (the highest transaction price on the previous day) of GP buyer 2. Recall that GP buyer 1 was a risk-taker using `PMinBid` the most when no other player in the market had learning ability. However, in the presence of another buyer with GP learning ability, GP buyer 1 became a truth-teller. Why did GP buyer 1 make this switch? One possible reason is that `HTV` is a more profitable strategy than `PMinBid` against `PMax`.

We then examined the data and found that `PMinBid` vs. `PMax` would give GP buyer 1 a profit of 20.5 and GP buyer 2 a profit of 29.5. `PMinBid` is actually a better strategy than `HTV` for GP buyer 1 (but not for GP buyer 2). Why didn't GP buyer 1 learn the better strategy? We have two hypotheses:

- 300 generations did not provide enough time for GP to learn the `PMinBid` strategy under the presence of another GP buyer whose strategies were also evolving. In the other experiment where there was only one GP buyer in the market, the earliest time `PMinBid` appeared in the population was at generation 60. In some of the runs, the strategy was not seen until generation 230. Using a trial-and-error learning style, GP needed time to obtain information and improve. If the number of generations had been greater, GP buyer 1 might have learned the strategy. We will test the hypothesis in our future work.

- The market environment for GP buyer 1 to learn the `PMinBid` strategy was not provided. The simulation data showed that GP buyer 1 used `PMinBid` only when GP buyer 2 used `HTV` (i.e., truth-telling). When there is only one GP buyer in the market, all other buyers used `HTV` (i.e., truth-telling). Under such market environment, the GP buyer learned and used `PMinBid` the most to conduct auction. However, once the market has changed such that buyer 2 had the ability to learn and found `PMax` was a better strategy than truth-telling, `HTV` was seldom used. Without the required market environment, GP buyer 1 did not learn that `PMinBid` was a better strategy to use.

For the p-24 vs p-37 pair, 4 combinations appeared frequently:

`Min PMax HTV` vs. `PMax`

`Min PMinBid HTV` vs. `PMax`

`Min PMax HTV` vs. `PMinBid`

`Min PMinBid HTV` vs. `PMinBid`

Recall that when there was only one GP buyer in the market, `Min PMinBid HTV` was not a strategy used by GP buyer 1 when its population size was 10. It was only when the population size was increased to 50 that this more sophisticated strategy emerged. However, under the presence of another GP buyer, buyer 1 was able to learn this more profitable strategy under the population size of 10. This indicates that the competitive co-evolution dynamics between GP buyer 1 and GP buyer 2 has promoted the emergence of this strategy without an increase in population size.

When the population size was increased from 10 to 50, the use of p-24 and p-37 increased, while the use of p-17 and p-35 decreased. This indicates that the two more "intelligent" GP buyers learned to use a more profitable strategy to conduct the auction. This co-evolution dynamics produced a win-win result as both gained more by using the p-24 and p-37 strategies more often in the auctions. However, this was not the case for the p-17 vs. p-35 pair as GP

buyer 2's gain was at the expense of GP buyer 1. As mentioned previously, the co-evolution dynamics led GP buyer 2 to use `PMax` more often as it resulted in a higher profit than did truth-telling. Without the presence of GP buyer 2's `HTV`, GP buyer 1 did not learn the more profitable `PMinBid` and used `HTV` which resulted in a lower profit.

To sum up, with the presence of another GP buyer in the market, GP buyer 1 changed its strategy in conducting auctions in the following ways:

1. By switching from `PMinBid` to the less profitable truth-telling strategy.
2. Capable of learning the more sophisticated and profitable `Min PMinBid HTV` strategies.

The aggregated result of the 2 changes was positive and the profit of GP buyer 1 increased. Meanwhile, by switching from truth-telling to being a GP learner, buyer 2 also gained more profit from the co-evolution of the two GP buyers. When the intelligence of both GP buyers was increased, they learned to use the more profitable strategies more often and both gained more profits.

8. Concluding Remarks

The behavior of a DA market is the result of the aggregated behavior of multiple traders. Due to the interdependence of the traders' actions, it is not easy to isolate or analyze any individual trader's auction strategies based on the market's overall performance. While the human subject experiment technology has been developed to analyze emerged market phenomena, the representation of human cognition is an unsettled issue. ABM provides an alternative to conduct market simulation in a controlled environment. The results of the ABM simulation provide a basis that can be validated by human subject experiments.

This research conducts DA market simulations on an ABM platform. By using GP to implement the bounded rationality of traders' decision making, we were able to study the macro market performance changes under traders with different degrees of rationality. Meanwhile, the individual GP buyer's strategies and the co-evolution of multiple GP buyers' strategies were analyzed.

In terms of the macro market performance, we observed that the DA market efficiency increases when the individual trader's intelligence increases. However, increasing the number of intelligent traders in the market led to a decline in market efficiency. This is because the more intelligent GP trader developed strategies to collect extra profits that did not conflict with others' profits. On the contrary, under multiple GP traders with learning ability, the market became more competitive. As a result, the GP traders developed aggressive strategies that damaged other traders' profits. The overall market efficiency was reduced.

The learning behavior of the individual GP trader differs when the market environment is different. When the market is stable (i.e., all other traders

are truth-tellers), the GP trader learns more simple rules that require less risk to gain more profit than that gained by being a truth-teller. When another GP trader joined the market, however, it was found that the market became dynamic in that both GP traders revised their strategies constantly to outdo each other. The co-evolution of the two GP traders' strategies generated both positive and negative impacts on the two traders. In one instance, both GP traders co-operated and applied strategies that benefited each other. In another instance, one GP trader applied strategies that prevented the other GP trader from learning more profitable strategies to protect his own profit. The third instance was where one GP trader learned more sophisticated strategies in this dynamic environment, which he did not learn under the stable environment.

All of the observed market performance and individual traders learning behavior make intuitive sense. Under the ABM platform, GP agents demonstrated human-like rationality in decision making. We plan to conduct human subject experiments to validate these results in the near future.

Acknowledgment

A preliminary draft of this paper was presented at the GPTP-2008 – Genetic Programming Theory and Practice (GPTP) Workshop, Ann Arbor, Michigan May 15-17, 2008. The authors are grateful for the comments received from the participants at the conference, especially, John Holland and Robert Reynolds. The paper is revised in light of two very helpful referee reports provided by B. Worzel, M. Kim, Y. Becker, P. Fei and U-M O'Reilly to whom the authors are grateful. The research support in the form of NSC grant No. NSC. 95-2415-H-004-002-MY3 is also gratefully acknowledged.

References

Arthur, W. Brian (1993). On designing economic agents that behave like human agents. *Journal of Evolutionary Economics*, 3:1–22.

Chen, Shu-Heng and Tai, Chung-Ching (2003). Trading restrictions, price dynamics and allocative efficiency in double auction markets: an analysis based on agent-based modeling and simulations. *Advances in Complex Systems*, 6(3):283 – 302.

Cliff, Dave and Bruten, Janet (1997). Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets. Technical Report HP-97-141, HP Technical Report.

Duffy, John (2006). Agent-based models and human subject experiments. In Tesfatsion, Leigh and Judd, Kenneth L., editors, *Handbook of Computational Economics*, chapter 19, pages 949 – 1011. Elsevier.

Gode, Dhananjay K. and Sunder, Shyam (1993). Allocative efficiency of markets with zero-intelligence traders: markets as a partial substitute for individual rationality. *Journal of Political Economy*, 101:119 – 137.

Koza, John R. (1992). Genetic evolution and co-evolution of game strategies. In *International Conference on Game Theory and Its Applications*.

Rust, John, Miller, John, and Palmer, Richard (1993). Behavior of trading automata in a computerized double auction market. In Friedmand, D. and Rust, John, editors, *The Double Auction Market: Institutions, Theories and Evidence*, pages 155 – 198. Addison-Wesley.

Simon, Herbert A. (1997). Behavioral economics and bounded rationality. In Simon, Herbert A., editor, *Models of Bounded Rationality*, pages 267 – 298. MIT Press.

Chapter 14

PROFILING SYMBOLIC REGRESSION-CLASSIFICATION

Michael F. Korns[1] and Loryfel Nunez[1]

[1]*Investment Science Corporation, 1 Plum Hollow, Henderson, Nevada 89052 USA*

Abstract This chapter performance-profiles a composite method of symbolic regression-classification, combining standard genetic programming with abstract grammar techniques, particle swarm optimization, differential evolution, context aware crossover, and age-layered populations. In two previous papers we experimented with combining high performance techniques from the literature to produce a large scale symbolic regression-classification system. In this chapter we briefly describe the current combination of techniques and make the assertion that Symbolic Regression-Classification (via Genetic Programming) is now ready for some industrial strength applications. This aggressive assertion is supported with performance metrics on multiple large scale problems which are scored on testing data which is always distinct from the training data. Each problem contains ten thousand fitness cases (rows) and problems vary in complexity from one to thirty independent variables (columns). In each problem, the dependent variable is generated by a test expression which varies in complexity from simple linear to complex non-linear often including conditionals (if-then-else clauses) with random noise starting at zero percent and rising up to forty percent.

Keywords: artificial intelligence, genetic programming, particle swarm optimization, differential evolution, portfolio selection, data mining, formal grammars, quantitative portfolio management

1. Introduction

This is the continuing story of the issues encountered by Investment Science Corporation in using composite genetic programming techniques to construct a large-scale, time-constrained symbolic regression-classification tool for use with financial data.

In two previous papers (Korns, 2006; Korns, 2007) our pursuit of industrial scale performance with large-scale, time-constrained symbolic regression

problems, required us to reexamine many commonly held beliefs and, of necessity, to borrow a number of techniques from disparate schools of genetic programming and "recombine" them in ways not normally seen in the published literature. We continue this tradition in this current paper, building a symbolic regression-classification tool combining the following disparate techniques from the genetic and evolutionary programming literature:

- Standard tree-based genetic programming
- Vertical slicing and out-of-sample scoring during training
- Grammar template genetic programming
- Abstract grammars utilizing swarm intelligence
- Context aware cross over
- Age-layered populations
- Random noise terms for learning asymmetric noise
- Bagging

For purposes of comparison, all results in this paper were achieved on two workstation computers, specifically an Intel©Core 2 Duo Processor T7200 (2.00GHz/667MHz/4MB) and a Dual-Core AMD Opteron©Processor 8214 (2.21GHz), running our Analytic Information Server software generating Lisp agents that compile to use the on-board Intel registers and on-chip vector processing capabilities so as to maximize execution speed, whose details can be found at *www.korns.com/Document_Lisp Language_Guide.html*.

Fitness Measure

Standard regression techniques often utilize least squares error (LSE) as a fitness measure. In our case we normalize by dividing LSE by the standard deviation of "Y" (dependent variable). This normalization allows us to meaningfully compare the normalized least squared error (NLSE) between different problems.

Of special interest is combining fitness functions to support both symbolic regression and classification of common stocks into long and short candidates. Specifically we would like to measure how successful we are at predicting the future top 10% best performers (*long candidates*) and the future 10% worst performers (*short candidates*) (Korns, 2007).

Briefly, let the dependent variable, Y, be the future profits of a set of securities, and the variable, EY, be the *estimates* of Y. If we were prescient, we could automatically select the best future performers *actualBestLongs, ABL,* and worst

future performers *actualBestShorts, ABS,* by sorting on Y and selecting an equally weighted set of the top and bottom 10%. Since we are not prescient, we can only select the best future estimated performers *estimatedBestLongs, EBL,* and estimated worst future performers *estimatedBestShorts, EBS,* by sorting on EY and selecting an equally weighted set of the top and bottom 10%. If we let the function *avgy* represent the average y over the specified set of fitness cases, then clearly the following will always be the case.

```
-1 <= ((avgy(EBL) - avgy(EBS)) / (avgy(ABL) - avgy(ABS))) <= 1
```

We can construct a fitness measure known as tail classification error, TCE, such that

```
TCE = ((1 - ((avgy(EBL) - avgy(EBS)) / (avgy(ABL) - avgy(ABS)))) / 2)
```

and therefore

```
0 <= TCE <= 1
```

A situation where TCE < .5 indicates we are making money speculating on our short and long candidates. Obviously 0 is a perfect score (we might as well have been prescient) and 1 is a perfectly imperfect score (other traders should do the opposite of what we do). Clearly, considering our financial motivation, we are interested in achieving superior regression fitness measures; but, we are also interested in superior classification. In fact, even if the regression fitness (NLSE) is poor but the classification (TCE) is good, we can still have an advantage, in the financial markets, with our symbolic regression-classification tool.

Since both the TCE and NLSE fitness measures are normalized, we can make standard interpretations of results across a wide range of experiments. In the case of NLSE, any score of .3 or less is very good (meaning the average least squared error is less than 30% of the standard deviation of Y), while a score of less than .5 is okay, NLSE scores greater than .5 indicate increasingly poor regression results. Our system automatically averages the estimates of the ten top champions (bagging) whenever the training NLSE of the top champion is greater than .5. Finally, a TCE score of less than .2 is excellent. A TCE score of less than .2 is good; while, a TCE of .5 or greater is poor.

Vertical Slicing

We make use of an out-of-sample testing procedure we call *vertical slicing*, wherein the rows in the training matrix X are sorted in ascending order by the dependent values, Y. Then the sorted rows in X are subdivided into S *vertical slices* by selecting every S-th row to be in each vertical slice. Thus the first vertical slice is the set of training rows as follows *X[0], X[S], X[2*S], ...* . We train on a single vertical slice, but test on all vertical slices (Korns, 2006).

Since Y represents the *behavior* of the system to be learned, sorting X by Y insures that each vertical slice contains training examples equally distributed across the range of behaviors of the system. For complete training coverage,

every epoch we randomly select a different vertical slice as the training subset while still scoring fitness across every fitness example in X.

Vertical slicing reduces training time (which in multiple regression and swarm grammars can be time consuming); while simultaneously reducing over fitting by scoring fitness over all slices (out-of-sample testing).

Abstract Grammar

Recently, informal and formal grammars have been used in genetic programming to enhance the representation and the efficiency of a number of applications including symbolic regression – see overviews in (O'Neill and Ryan, 2003) and (Poli et al., 2008).

Our system implements a hybrid combination of tree-based GP and formal grammars where the head of each sublist is a grammar rule with polymorphic methods for mutation, crossover, etc. Different grammar rules communicate with each other by message passing. We use standard mutation and crossover operations (Koza, 1992) and support both simple regression and multiple regression, as follows: *regress(expression)*; and *regress(expression,...,expression)*.

Our numeric expressions are JavaScript-like containing the variables *x0* through *xm* (where *m* is the number of columns in the regression problem) and real constants such as *2.45 or -34.687*, with the following binary and unary operators +, -, /, %, *, <, <=, ==, ! =, >=, >, expt, max, min, abs, cos, cosh, cube, exp, log, sin, sinh, sqroot, square, tan, tanh, and the ternary conditional expression operator if (...) then ... else ...;

In (Korns, 2006), we implemented both the production and recognition side of our grammars. Later in (Korns, 2007) we developed an abstract grammar which is evaluated using swarm intelligence and provides excellent fine grain control during evolution.

The enhancement of abstract real constants *c0* through *ck* (where *k* is the number of unique abstract real constants in the expression) allows more fine grain control over the evolution of optimal real numbers during the GP process.

For instance, the following concrete expression *regress(3.4*sin(x3))*, when evaluated, has a fitness score based upon regressing 3.4 times the sine of variable three. However, the following abstract expression *regress(c0*sin(x3))*, which must be evaluated in a swarm agent, has a fitness score based upon the swarm agent's choice of optimal real constant for *c0*. Our experience is that swarm intelligence is an excellent method of optimizing specific real constants.

Additionally, the enhancement of abstract variables *v0* through *vj* (where *j* is the number of unique abstract variables in the expression) allows more fine grain control when optimizing formulas with a specific grammatical format.

For instance, the following abstract expression *regress(c0*sin(v0))*, which must be evaluated in a swarm agent, has a fitness score based upon the swarm

agent's choice of actual real constant for *c0* and the swarm's choice of actual concrete variable *v0*. An example of an optimized champion from the swarm agent might be *regress(-.416*sin(x10))*. One is always assured that the final swarm optimized champion will be in a form compatible with the abstract grammar expression supplied. The possibilities for addressing bloat are obvious.

This year, as an extension of our previous experiments with abstract grammar, we introduce abstract random noise terms *e0* through *ek* for learning asymmetric noise (Schmidt and Lipson, 2007). For instance, the following abstract expression *regress(c0*sin(v0+e0))*, which must be evaluated in a swarm agent, is evaluated iteratively an additional number of times with random values between -1 and +1 replacing the *e0* noise term. The fitness score is based upon the swarm agent's choice of actual real constant for *c0* and the choice of actual variable for *v0*, while the noise term *e0* is randomly fluctuated between -1 and +1 as described in greater detail in (Schmidt and Lipson, 2007).

By using an abstract expression grammar with imbedded swarm optimization, we achieve more fine-grained control, of numeric constant optimization, expression bloat, and learning asymmetric noise.

Context-aware Crossover

In (Majeed and Ryan, 2006) an extension of standard GP crossover was devised. In standard GP crossover (Koza, 1992), a randomly chosen snip of genetic material from the father s-expression is substituted into the mother s-expression in a random location. In context-aware crossover, a randomly chosen snip of genetic material from the father s-expression is substituted into the mother s-expression *at all possible valid locations*. Where standard crossover produces one child per operation, context-aware crossover can produce many children *depending upon the context* (Korns, 2007).

We further extended context-aware crossover such that *all possible valid* snips of genetic material from both the mother and father are substituted into both parents *at all possible valid locations*. Each of the many offspring are evaluated with only the survivors being added to the pool. We add our extended context-aware crossover to all GP runs in our system with a increasing probability with each additional generation.

Context-aware crossover holds-forth the promise of greater coverage of the local search space, as defined by the candidate s-expressions' roots and branches, and, therefore, a greater control of the evolutionary search at a fine grain level.

Age-Layered Populations

In (Hornby, 2006) a technique is introduced, known as *aged-layered population structure* (ALPS), devised to minimize premature population convergence.

ALPSs evolve populations in several significant ways. New random populations are generated at irregular intervals over the duration of the evolutionary process. Populations are segmented by the age of the individuals with evolutionary competition restricted to individuals within roughly the same age cohort (young individuals are not allowed to be swamped by more mature individuals). At irregular intervals, younger champions are promoted into populations allowing competition with older individuals (Hornby, 2006).

ALPS differs from a typical evolutionary process by segregating individuals into different age-layers and by regularly introducing new, randomly generated individuals in the youngest layers. The introduction of randomly generated individuals at regular intervals results in an evolutionary process that is never completely converged and is always exploring new parts of the fitness landscape. By restricting the age of competitors and breeding within an age cohort, younger individuals are able to develop without being swamped by older ones. Analysis of the behavior of ALPS finds that individuals that are randomly generated mid-way through a run are able to move the population out of mediocre local optima to better parts of the fitness landscape if they are allowed to develop in their youth, free from unfair competition by more mature adults.

Experimental Setup

Our goal is to profile the performance of our *genetic symbolic regression-classification machine* (GSM) on a series of symbolic regression-classification problems. All of our symbolic regression-classification problems consist of a training phase and a distinct testing phase. In the training phase, an (N x M) real number matrix, X, is filled with random numbers[1]. Then a training model, f(x), is selected to create a real number N-dimensional vector of dependent variables, Y, as follows:

```
Y = f(X) + noise;   vector equation
```

Training models vary from simple linear all the way to complex nonlinear. For example:

```
y = 1.57 + (1.57*x0) - (39.34*x1);   simple linear
y = ((1.57*x0)%(39.34*x1))
     + (if (log(x0) == x2) {sin(2.13*x2)}
    else {((39.34*x1)%(2.13*x2))});   complex nonlinear
```

In (Korns, 2007) we published nine base training models which vary from simple linear to complex nonlinear. In this paper we generate numerous training models at random. These training models vary from simple linear to complex

[1]N refers to number of rows in the matrix (always 10,000 rows unless otherwise stated). M refers to the number of columns (columns vary from 1 to 30 depending upon test difficulty). The range of the random numbers is from -50 to +50 unless otherwise stated.

nonlinear[2]. Furthermore, our problem set varies from one column problems to more difficult thirty column problems, and from zero random noise to problems with 40% random noise, which is generated as follows:

```
y = (y * 0.8) + (y * random(0.4));
```

The output of the training phase is a champion estimator model, *ef*, which, when evaluated on the training matrix, generates an estimated N vector, EY:

```
EY = ef(X) + noise;   estimator model
```

In the testing phase, another (N x M) real number matrix, TX, is filled with random numbers. Then the original training model, f(x), is evaluated on TX to create a real number N vector of dependent testing variables, TY. Then the champion estimator model, ef, is evaluated on the testing matrix, TX, and the normalized least squares error (NLSE) difference between EY and TY is the final out-of-sample testing score.

For each of the numerous experimental problems attempted for this profiling study, we collected the following important data: Unique Problem Identifier; Training generations to completion; Training noise; Training minutes to completion; Training normalized least squared error (NLSE); Testing normalized least squared error (NLSE); Testing classification error (TCE); Champion estimator model; and the Training model.

Results on Nine Base Problems

The results of training on the nine base training models on 10,000 rows and twenty columns with 40% random noise and only 20 generations allowed, are shown in Table 14-1.

Fortunately, training time is very reasonable given the difficulty of some of the problems and the limited number of training generations allowed. In general, average percent error performance is poor with the *linear* and *cubic* problems showing the best performance. Extreme differences between training error and testing error in the *mixed* and *ratio* problems suggest over-fitting. Surprisingly, long and short classification is fairly robust in most cases with the exception of the *hidden*, *ratio* and *hyper* test cases. If we were to run a market neutral hedge on hypothetical markets, driven by these nine test models, we would have made money in all but one of the markets, made a little money in the markets driven by the *hidden* and *hyper* models, broken even in the market driven by the *ratio* model, and made excellent money in all other markets.

The salient observation is the relative ease of classification compared to regression even in problems with this much noise. In five of the test cases, testing NLSE is either close to or exceeds the standard deviation of Y (not very good); however, in six of the test cases classification is below 20

[2]The complete set of randomly generated training models are described in detail on www.korns.com.

Table 14-1. Result for 10,000 rows by 20 columns with Random Noise. The columns are: *Test:* The name of the test case; *Minutes:* The number of minutes required for training; *Train-Error:* The average percent error score for the training data; *Test-Error:* The average percent error score for the testing data; and *Classify:* The classification score for the testing data.

Test	*Minutes*	*Train-Error*	*Test-Error*	*Classify*
cross	9	0.80	0.80	0.19
cubic	10	0.11	0.11	0.00
hyper	9	0.96	0.96	0.36
elipse	12	0.45	0.46	0.05
hidden	10	0.99	0.99	0.45
linear	10	0.11	0.11	0.00
mixed	12	0.69	1.85	0.07
ratio	26	0.95	1.18	0.46
cyclic	8	0.39	0.91	0.18

Table 14-2. Result For 10,000 rows by 5 columns no Random Noise. The columns are the same as in Table 14-1.

Test	*Minutes*	*Train-Error*	*Test-Error*	*Classify*
cross	107	0.37	0.39	0.02
cubic	0	0.00	0.00	0.00
hyper	369	0.00	0.00	0.00
elipse	0	0.00	0.00	0.00
hidden	3	0.00	0.05	0.00
linear	0	0.01	0.01	0.00
mixed	123	0.24	1.65	0.13
ratio	6	0.03	1.05	0.50
cyclic	4	0.04	0.14	0.06

The obvious comparison with the above difficult problems would be to try easier problems by eliminating the random noise, reducing the number of columns, and increasing the number of training generations allowed. The results of training on the nine base training models on 10,000 rows and *five columns* with *no random noise* and up to *200* generations allowed, are shown in Table 14-2.

Fortunately, reducing the problem difficulty greatly improves the results. In general, average percent error performance is now very good. Extreme differences between training error and testing error in the *mixed* and *ratio* problems

suggest over-fitting. Long and short classification is excellent in most cases with the exception of the *ratio* test case.

Now let us examine the performance of the GSM tool in the many problems whose difficulty falls in between the two extremes shown above. In the next few sections, we will profile the behavior of the tool on a wide range of randomly generated problems.

Results on Advanced Problems. Using the system's production grammar features, we generated thousands of symbolic regression problems ranging from 1 to 30 columns in complexity, from 0% to 40% noise, and containing simple root expressions as well as more difficult modal expressions. We gave each problem 20 generations in which to evolve a solution.

We classify the problems according to the level of difficulty in the column expressions generated for each symbolic regression problem. *Root* problems do not contain conditionals (if then else clauses). *Modal* problems may contain conditionals.

- Example of a one-column root expression:

```
y = (11.3665735467+(16.94203837131*(exp(x0) % abs(x0))));
```

- Example of a one-column modal expression:

```
y = (-20.29753816981+(12.1706446781*(if ((tanh(x0) - x0) >=
(abs(x0) - sqrt(x0))) ((x0*x0*x0) + (x0*x0))
else (exp(x0) - sign(x0)))));
```

There are thousands of separate, independent symbolic regressions tests reported in this paper. For each problem type (Root or Modal) there are 12 unique column-noise combinations. There are a total of 24 test cases all in all.

A test is considered excellent if its NLSE is less than .31 and reasonably good if its NLSE is less than .51. Additionally, a test is considered excellent if its TCE is less than .21 and reasonably good if its TCE is less than .31.

The results of training on the randomly generated training models on 10,000 rows and from 1 to 30 columns with 0 to 40 percent noise are shown in Table 14-3.

For trivial single-column problems, almost all tests show almost perfect classification scores even for problems with 40% noise. As we increase the number of columns, we lose our accuracy as seen from the increasing NLSE score. But even in our most difficult problems we are achieving over 70% excellent classification scores. This is unprecedented.

Effects Number of Column and Noise Levels. We randomly generated tests for the different test cases as seen in Table 14-3. We generated tests for the following number of columns: 1, 10, 20, and 30, with the following noise levels: 0%, 20%, and 40% for simple root expressions and for more difficult modal expressions.

Table 14-3. Result for 10,000 rows by 1-30 columns with 0-40% Random Noise. The top of the results are for the Root tests and the second half is for the Modal tests, as indicated in the left-most column. The rest of the columns are: *Cols:* The number of columns of data $(1, 10, 20, 30)$; and *Noise:* The amount of noise added $(0\%, 20\%, 40\%)$; *Count:* Total number of such tests; *TrainNLSE:* Average training error for all the tests; *TestNLSE:* Average testing error for all tests; *TCE:* Average classify score for all tests; *% GoodReg:* Percent of tests with a test NLSE less than 0.31; and *% GoodClass:* Percent of tests with a TCE Score less than 0.21.

	Cols	Noise	Count	Train NLSE	Test NLSE	TCE	%Good Reg	%Good Class
R	**1**	0%	23	0.016	0.015	0.0007	100.00	100.00
O		20%	25	0.131	0.132	0.032	92.00	98.61
O		40%	25	0.158	0.161	0.014	96.00	95.83
T	**10**	0%	130	0.285	0.286	0.036	65.38	93.85
		20%	107	0.309	0.294	0.062	68.22	88.79
		40%	86	0.369	0.370	0.080	63.95	87.21
	20	0%	66	0.422	0.394	0.074	50.00	90.91
		20%	90	0.460	0.450	0.084	43.33	84.44
		40%	37	0.463	0.423	0.050	45.95	91.89
	30	0%	55	0.604	0.608	0.118	23.64	74.55
		20%	52	0.625	0.560	0.099	26.92	84.62
		40%	51	0.616	0.560	0.095	17.65	84.31
M	**1**	0%	47	0.043	0.034	0.003	97.87	100.00
O		20%	72	0.120	0.121	0.021	94.44	98.61
D		40%	48	0.243	0.246	0.054	77.08	95.83
A	**10**	0%	88	0.405	0.318	0.044	62.50	90.91
L		20%	117	0.439	0.390	0.048	56.41	90.60
		40%	39	0.466	0.462	0.036	46.15	97.44
	20	0%	68	0.444	0.455	0.062	44.12	92.65
		20%	69	0.630	0.577	0.093	30.43	86.96
		40%	85	0.552	0.534	0.059	37.65	89.41
	30	0%	80	0.649	0.579	0.080	30.00	86.25
		20%	65	0.690	0.658	0.101	15.38	83.08
		40%	117	0.730	0.676	0.118	17.09	75.21

On first glance at Table 14-3 and at Figures 14-1 and 14-2, the classification score (*TCE*) and the regression score (*TCE*) get worse as the number of columns and the level of noise increase. To test the separate and mutual effects of the number of columns, and noise levels on the classification score (*TCE*) and the Test error (*NLSE*) , we performed the ANOVA on our test results at 0.01 level of significance.

For the classification score (*TCE*), the ANOVA results are the same for both root and modal problems. The results show that the number of columns and noise level separately have significant F-values, with the noise level posing a

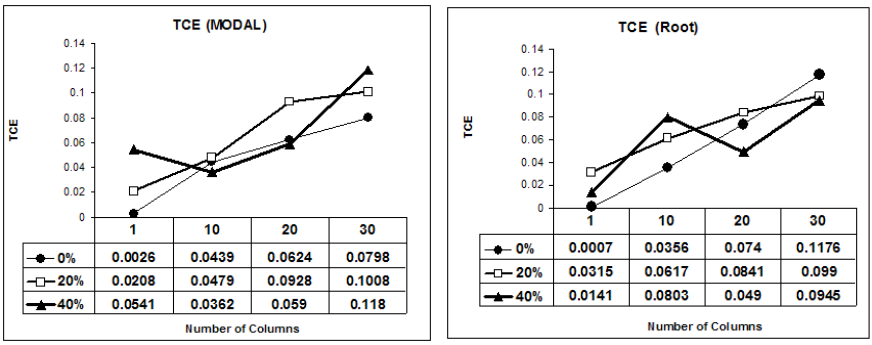


Figure 14-1. TCE Score Results for Root and Modal Tests, for 0%, 20%, and 40% noise levels, and 1, 10, 20 and 40 columns.

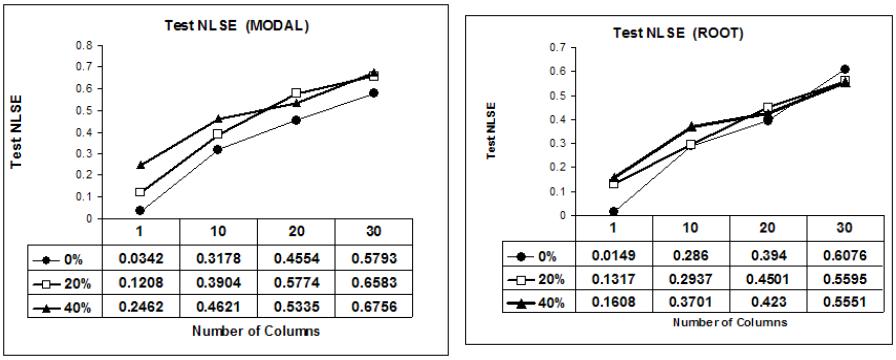


Figure 14-2. Test NLSE Score Results for Root and Modal Tests for 0%, 20%, and 40% noise levels, and 1, 10, 20 and 40 columns.

higher F-score than the number of columns. The ANOVA results for the interaction between noise level and number of columns is not statistically significant.

This means that increasing number of columns or the noise level alone worsens the TCE. Increasing the noise level has more effect in worsening the TCE than increasing the number of columns. Surprisingly, there is no statistical confirmation that an increase in the number of columns with a corresponding increase in noise level will significantly worsen the TCE.

For the regression score (*Test NLSE*), using the F-test at 0.01 level of significance, the ANOVA results for the interaction between noise levels and number of columns show very high F-values for both root and modal problems. However, for root problems, the number of columns or the noise level separately do not pose significant F-values. For modal problems, only the number of columns pose a significant F-value.

This shows that ANOVA results provide no statistically significant confirmation that there will a corresponding increase in the NLSE when we increase the

Table 14-4. 30-column problems with varying Noise Levels. The column headings are as follows: *Test:* Number of Tests; *20Gens:* Average Test NLSE or TCE for 20 generations; *200Gens:* Average Test NLSE or TCE for 200 generations; *Difference:* Test NLSE or TCE in 200 Generations; and subtracted by Test NLSE or TCE in 20 Generations.

			TEST NLSE			TCE		
		Tests	20 Gens	200 Gens	Diff	20 Gens	200 Gens	Diff
Modal	0%	32	0.870	0.632	0.238	0.297	0.125	0.171
	20%	33	0.785	0.714	0.070	0.125	0.146	-0.021
	40%	31	0.661	0.601	0.060	0.112	0.110	0.002
Root	0%	34	0.637	0.615	0.022	0.134	0.332	-0.197
	20%	26	0.716	0.578	0.139	0.138	0.115	0.022
	40%	19	0.568	0.524	0.044	0.083	0.136	-0.053

number of columns or the noise level separately for root problems. This result is different for modal problems where an increase in the number of columns alone would significantly worsen the NLSE. For both root and modal problems, it is an increase in the number of columns coupled with a corresponding increase in the noise level that worsens NLSE as seen from the resulting high F-scores for the noise-column interaction.

Effects Increasing the Number of Training Generations. We have increased the number of generations from 20 to 200 in our 30-column tests to see if the evolved solution is more accurate.

From the results in Table 14-4, Test NLSE Results are improved by increasing the number of generations for both modal and root problems. However, increasing the number of generations do not necessarily improve TCE results. In fact, in some cases due to over fitting, TCE scores after 200 generations are worse.

An additional uncompleted test is currently underway. The system has been given a complicated modal problem with five variables (columns), no noise, and 20,000 generations to train. At the time of this publication we see continuous improvement in the training NLSE. At 20 generations the training NLSE was .75, by 100 generations training NLSE had dropped to .65, by 600 generations training NLSE had fallen to .64, and at 7,000 generations the training NLSE is now down to .37.

There does not appear to be any hard barrier to continuous training improvement in this system (we believe this to be due to the ALPS strategy). However, we will not know until generation 20,000 whether or not the system has over fit (as happened in some cases above after only 200 generations).

Summary

Genetic Programming, from a corporate perspective, is ready for industrial use on *some* large scale, time constrained symbolic regression-classification problems. Adapting the latest research results, has created a symbolic regression tool whose efficiency is exciting.

While there is no evidence to suggest that the eight techniques included in this system are the absolute best, it is a credit to the scientists pioneering Genetic Programming that at least this combination of techniques has produced a system, which for hundreds of randomly generated difficult modal problems with 30 columns and 40% noise, we obtained an excellent classification score in 75% of the test cases.

Financial institutional interest in the field is growing while pure research continues at an aggressive pace. Further applied research in this field is absolutely warranted. We are using our *genetic symbolic regression-classification machine* (GSM) in the financial domain. But as new techniques are added and current ones improved, we believe that GSM has evolved to be a domain-independent tool that can provide superior regression and classification results for industrial scale symbolic regression problems.

Additional research separating the effects of each of the eight techniques on training time, and testing NLSE/TCE is necessary. Currently we need a more detailed understanding of the individual effects of each of these techniques and of their effects in specific combinations. Furthermore, we need more research on the effects of longer training times (20,000 generations and more) on system convergence and over fitting.

Clearly we need to experiment with techniques which will improve our performance on the modal test cases. The primary area for future research involves experimenting with statistical other types of analysis to help build conditional WFFs for difficult multi-modal problems.

References

Hornby, Gregory S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 815–822, Seattle, Washington, USA. ACM Press.

Korns, Michael F. (2006). Large-scale, time-constrained symbolic regression. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Pro-*

gramming Theory and Practice IV, volume 5 of *Genetic and Evolutionary Computation*, chapter 16, pages –. Springer, Ann Arbor.

Korns, Michael F. (2007). Large-scale, time-constrained symbolic regression-classification. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 4, pages 53–68. Springer, Ann Arbor.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Majeed, Hammad and Ryan, Conor (2006). Using context-aware crossover to improve the performance of GP. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 847–854, Seattle, Washington, USA. ACM Press.

O'Neill, Michael and Ryan, Conor (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Dordrecht Netherlands.

Poli, Riccardo, Langdon, William B., and McPhee, Nicholas Freitag (2008). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

Schmidt, Michael D. and Lipson, Hod (2007). Learning noise. In Thierens, Dirk, Beyer, Hans-Georg, Bongard, Josh, Branke, Jurgen, Clark, John Andrew, Cliff, Dave, Congdon, Clare Bates, Deb, Kalyanmoy, Doerr, Benjamin, Kovacs, Tim, Kumar, Sanjeev, Miller, Julian F., Moore, Jason, Neumann, Frank, Pelikan, Martin, Poli, Riccardo, Sastry, Kumara, Stanley, Kenneth Owen, Stutzle, Thomas, Watson, Richard A, and Wegener, Ingo, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1680–1685, London. ACM Press.

Chapter 15

ACCELERATING GENETIC PROGRAMMING THROUGH GRAPHICS PROCESSING UNITS

Wolfgang Banzhaf[1], Simon Harding[1], William B. Langdon [2] and Garnett Wilson [1]

[1]*Dept. of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada;*
[2]*Dept. of Mathematical Sciences, Essex University, UK.*

Abstract Graphics Processing Units (GPUs) are in the process of becoming a major source of computational power for numerical applications. Originally designed for application of time-consuming graphics operations, GPUs are stream processors that implement the SIMD paradigm. The true degree of parallelism of GPUs is often hidden from the user, making programming even more flexible and convenient. In this chapter we survey Genetic Programming methods currently ported to GPUs.

Keywords: graphics processing units, parallel processing

1. Introduction

There is a notorious drag on Genetic Programming. Program search spaces are huge, and often difficult to navigate (Banzhaf et al., 1998; Langdon and Poli, 2002). Statistical measures for performance frequently require a large number of runs of GP systems in order to arrive at significantly safe statements. Finally, even running a single program might cause grief, due to the potentially large number of fitness cases required for a GP system to evolve a reasonable model of the underlying problem. In the extreme case, fitness cases have to be drawn from a probability distribution, and there will never be the same fitness case shown to the system again. To make things worse, measurements or simulations used to provide the data for a single fitness case evaluation might run from microseconds to days.

A simple calculation can easily demonstrate the amount of processing power required. Suppose we use a standard tree GP system, with each node being evaluated within $10^{-6}s$. If we assume that the problem is realistically difficult

for individual programs to regularly reach a depth limit of 17 using binary function nodes, an individual program will evaluate in approximately $0.131\ s$. Taking part of that size (not every program will be of maximum depth), 10^5 nodes, and multiplying with standard parameters for a GP run (number of fitness cases 10^3, population size 10^3, generations until end of run 10^3, and number of runs for statistical purposes 10^2), we end up with a runtime of $10^{10} s$, or 317 yrs. Even going to a billion node evaluations per second, which is certainly on the optimistic side, will require us to run the experiment for 116 $days$. So realistically, only experiments which can knock off a factor of 10^3 can reasonably be expected to be done.

Clearly, these types of considerations lead to restrictions on the types of problems presently being subjected to Genetic Programming, as well as to evaluation limitations in terms of number of fitness samples, or length of programs. This, in turn, will change the evolvability and quality of solution, or, in the former case, even prohibit certain problems from being addressed by GP.

With GPUs, the situation is bound to change fundamentally. "As of 2007, the fastest PC processors perform over 30 GFLOPS. GPUs in PCs are considerably more powerful in pure FLOPS. For example, in the GeForce 8 Series the NVIDIA 8800 Ultra performs around 576 GFLOPS on 128 processing elements. This equates to around 4.5 GFLOPS per element, compared with 2.75 per core for the Blue Gene/L supercomputer. It should be noted that the 8800 series performs only single precision calculations, and that while GPUs are highly efficient at calculations they are not as flexible as a general purpose CPU." (Wikipedia, 2008) With the advent of that kind of processing power, more problems become accessible, and now it is feasible for researchers to try a whole set of problems of a real-world application - rather than 1 or 2 demonstrations.

2. Various Sources of Speed-up for Genetic Programming

Problems of resource demand have always been with GP, but it appears as if the GP community has been pretty successful in circumventing the worst obstacles. It is clear that GP is very amenable to speedup through various methods. Here we'll just shortly outline a few of the sources for this potential.

First, there clearly are some factors that allow independent decomposition of tasks which lend themselves easily to parallelization:

1) Individual programs are evaluated using multiple independent fitness cases;

2) Populations consist of individuals which could be evaluated on independent hardware in parallel;

3) Repeated (independent) runs for producing appropriate statistical confidence levels can be executed simultaneously on different hardware.

Other factors do not lend themselves to easy parallelization because they require dependence, yet may also be exploited:

4) Evolutionary generations which build on each other

5) Program execution which usually requires a sequence of execution steps to be performed

6) Evaluation of single nodes which might be highly complex and serial processes

We think the above mentioned parallelization potential corresponds to the following causes:

1) Difficulty of the problem;

2) Population size chosen;

3) Statistical significance of results;

4) Speed of evolution with a given representation;

5) Size of the search space;

6) Complexity of the problem domain.

Issues (1) to (3) are more easy to address in a parallel computing environment while issues (4) to (6) are more difficult, but not impossible, to address.

There is probably no generic solution to problem (6), but accelerating the evaluation of single nodes is certainly possible, either directly (see below in the case of image filters) or indirectly by using proxy evaluation (Ziegler and Banzhaf, 2003). It is more difficult to address the dependency problems of (4) and (5), although attempts have been made to relax the sequentiality of programs, see, e.g. (Banzhaf and Lasarczyk, 2004). As for 4), an entire area of research has sprung up around more efficient and more easily evolvable representations, which has a bearing on the speed of evolution. But a radical approach like dissolving the sequence of generations is not possible, given the very nature of evolution, although some inroads can be made in dissolving the dependency, e.g. the success of steady-state algorithms or evaluation queueing.

In general, it can be expected that the hardware landscape is now sufficiently heterogeneous that each case will have another optimum for the expoitation of parallel resources. It remains to be seen how adaptive algorithms (forthcoming 2008) will be able to take advantage of these resources.

3. Classical Parallelization

Early work on parallelization of Genetic Programming already includes SIMD architecture approaches (Juille and Pollack, 1995) and a GP system written in FORTRAN running on a Cray super computer (Turton et al., 1996). Even earlier was work porting classifier systems / GAs to Thinking Machines SIMD architecture of the Connection Machine (CM) back in the 1980s (Robert-

son and Riolo, 1988) which even preceded the coevolution of parallel sorters by Hillis (Hillis, 1990).

Applications were the driving force for parallel approaches to GP (Oussaidene et al., 1996; Stoffel and Spector, 1996), no wonder given the restrictions on single runs discussed earlier. Koza at al. (Andre and Koza, 1996; Bennett III et al., 1999) popularised the use of transputer boards and later Beowulf workstation clusters where the population is split into separately evolving demes with limited emigration between compute nodes or workstations (Page et al., 1999). Multipopulation models were more systematically studied by (Fernández et al., 2003). New techniques became also available through progress with parallel hardware (Folino et al., 2003b; Folino et al., 2003a).

A number of groups demonstrated the use of the Internet for parallel GP, e.g., (Chong and Langdon, 1999; Gross et al., 2002; Folino et al., 2006). In these approaches the GP population can be literally spread over the globe. Alternatively JavaScript has been used to move interactive fitness evaluation to the user's own home but retain elements of a centralised population (Langdon, 2004). In recent times, cloud computing has been added to the options.

Others have used special purpose hardware. For example, (Eklund, 2003) used a simulator and was able to show how a linear machine code GP can be run very quickly on a field programmable gate array under VHDL to model sun spot data. FPGA were used already early on as acceleration tools for fitness evaluation (Koza et al., 1997). Martin employed a special C-compiler for FPGAs to run a GP system (Martin, 2001). However, FPGA architectures are intrinsically cumbersome to handle for an average programmer, and should be considered tools for the specialist, in contrast to Genetic Programming on GPUs.

One important consideration with every sort of parallel approach is the price one pays for hardware to accelerate computation. Manufacturers have exploited the need of scientists for computation for a long time. Cray and other parallel computers, including recent machines like IBM's BlueGene/L, carry a high pricetag.

Computing clusters, on the other hand, consist of commodity hardware and are therefore much cheaper. However, they also have high and continued operating costs, just as commercial parallel machines. Over the lifetime of even a small cluster, the operating costs can become several times that of the initial hardware investment (Feng et al., 2002). The costs come from a variety of sources, such as system adminstration, power, cooling and space. Being able to use a single desktop machine mitigates most of these costs.

In recent years, it has become recognized that GPUs provide much more economical means of achieving extremely parallel computation. To provide some indicative statistics, in mid-2007 the latest Intel CPU, the Core 2 Extreme QX6800, was capable of over 37 GFLOPS (INTEL, 2008). The CPU also had

a suggested retail price of $999 US at release, providing the consumer with a cost of approximately $27/GFLOP.

One of the latest benchmarked PC NVIDIA graphic cards, on the other hand, the GeForce 8800 GTX, provides 520 GFLOPS (NVIDIA, 2006) at an initial retail price of US $599 for $1.15/GFLOP. The NVIDIA 8800 has been used in much of the research presented in this chapter.

As one of the largest drivers of GPU power, computer gaming has emerged. Video game consoles are often sold at a loss in terms of hardware at product launch. This makes them a very economical choice for parallel power for parallel computation research. For instance, as early as 2005, the MS Xbox 360 claimed to provide 1 TFLOP (overall system performance, using both CPU and GPU computation) (XBOX, 2008). The most economical Xbox 360 package at that time sold for $299 US, providing a cost of $0.29/GFLOP. In this chapter, we describe how the Xbox 360 can be used to perform both general purpose computation, and GP.

4. The GPU platform and its potential

Graphics Processing Units are specialized stream processors, useful for rendering graphics applications. Typically, a GPU is able to perform graphics manipulations at a much higher speed than a general purpose CPU, since the graphics processor is specifically designed to handle certain primitive operations which occur frequently in graphics (game) applications. Internally, the GPU contains a number of small processors (see Figure 15-1) that are used to perform calculations on 3D vertex information and textures. A texture is a collection of pixels, in the form of a 2D image.

GPUs are constructed so that the simple vertex and shader processors work in parallel and in a pipeline fashion. The vertex processors calculate the 3D view, then the shader processors paint the model before it is displayed. Depending on the power of a GPU, the amount of parallel processors currently runs from 2 to 64 on both the vertex and the shader level. The fact that GPUs have the ability to perform restricted parallel processing has elicited considerable interest among researchers with applications requiring intensive parallel computation. Although the type of parallel processing used by GPUs, SIMD, is not the most general model of parallel computing, the sheer amount of raw processor power and the comparable low cost on graphics cards make it an attractive choice for a large number of applications, e.g. in scientific computing.

A GPU processes textures and outputs a vector of four floating point numbers for each texture element (texel) processed, traditionally corresponding to RGBA (red, green, blue, and alpha, for transparency) channels of a color. The two components of a GPU architecture that a user can control are the set of vertex processors and the set of pixel (or fragment) processors. An effect file, which

is a program to control the GPU, is divided into two parts corresponding to the architecture: a pixel shader and a vertex shader. The vertex shader program transforms input vertices based on camera position, and then each set of three resulting vertices computes a triangle from which pixel (fragment) output is generated and sent to the pixel processors. The shader program instructs the pixel shaders (processors) to "shade" each pixel in parallel and produce the final pixel with associated RGBA values for final output. Even though the latest GPUs (such as all those used this paper) use unified architectures, where the shader processors can handle vertex or pixel commands, the two functionalities are still separated when composing effect files.

While early GPUs tended to be severely restricted in the type and number of instructions they could execute, newer GPU architectures foresee the tendency to make freer use of the processing resources offered. Today, the number of instructions executable on a shader processor is not limited any more, and the type of instructions is broad, like all floating-point operations, instead of narrow. General purpose applications of GPUs (GPGPU) tend to take advantage of pixel shader programming rather than using the vertex processors, mainly because there are typically more pixel- than vertex-shaders on a GPU and the output of the pixel shaders is fed directly to memory (Harris, 2005). In terms of traditional data structures and execution, GPU textures are analogous to arrays, the shader program is like a Kernel program, and rendering effectively executes the program on array elements in parallel.

Programming a GPU

In this section we provide a brief overview of some of the general purpose computation toolkits for GPUs that are available. This is not an exhaustive list, but is intended to act as a guide to others. Generally speaking, the field is quickly advancing and readers are advised to carefully check the internet for the newest resources.

Sh and RapidMind: Sh is an open source project for accessing the GPU under C++ (RapidMind, 2008; LibSh Wiki, 2008). Many graphics cards are supported, and the system is platform independent. Many low level features can be accessed using Sh, however these require knowledge of the mechanisms used by the shaders. The Sh libraries provide typical matrix and vector manipulations, such as dot products and addition-multiplication operators. Sh has now been developed into a commercial product called RapidMind. A key benefit of the RapidMind toolkit is the ability to target other multicore platforms, such as the Cell/BE processor which is found in Playstation3. The usefulness of RapidMind for genetic programming has already been demonstrated for parallel evaluation of entire populations with more than 1 million individuals, all run

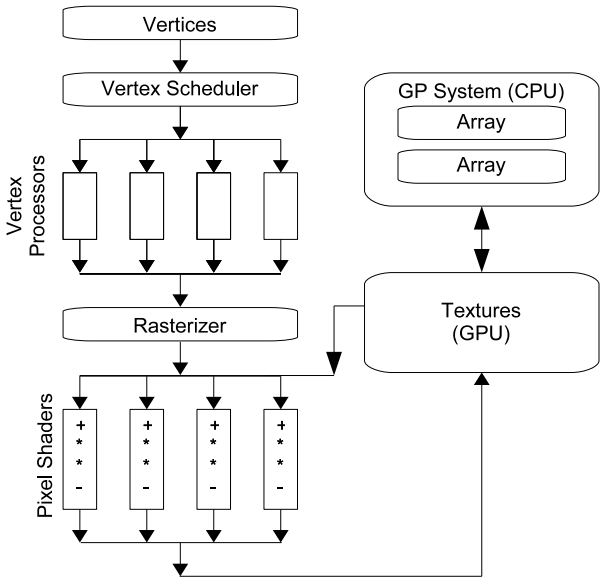


Figure 15-1. General sketch of GPU architecture and a GP implementation. Two sets of parallel stream processors are used to accelerate computation, vertex processors and pixel shaders. Only the second set of processors is used in these implementations.

in parallel (Langdon and Banzhaf, 2008), and we briefly mention its application here to Bioinformatics problems in Section 5.0.

Accelerator: Microsoft also has a prototype .Net assembly called Accelerator that provides access to the GPU via the DirectX interface (Tarditi et al., 2006). The system is completely abstracted from the GPU, and presents the end user with only arrays that can be operated on in parallel. Unfortunately, the system is only available for the Windows platform due to its reliance on DirectX. However, the assembly can be used from any .Net programming language.

Accelerator shares with RapidMind the design feature that operations are not performed on the data until the evaluated result is requested. This enables a certain degree of real time optimization, and reduces the computational load on the GPU. In particular, optimisation of common sub expressions will reduce the creation of temporary shaders and textures. The movement of data to and from the GPU can also be efficiently optimized, which reduces the impact of the relatively slow transfer of data out of the GPU. The compilation to the shader model occurs at run time, and hence can automatically make use of the different features available on the supported graphics cards.

We have previously used Accelerator for several different genetic programming tasks (Harding and Banzhaf, 2007a; Harding and Banzhaf, 2007b; Hard-

ing, 2008). In Section 5.0 of this paper we demonstrate the evolution of image filters using this toolkit.

CUDA and Cg: The GPU manufacturer NVIDIA also provides two toolkits for GPU programming. The Cg (C for graphics) toolkit allows one to develop shader programs for OpenGL or DirectX (NVIDIA, 2008; Mark et al., 2003). Cg provides extensions to the C language (such as data types) to make it suitable for GPU programming. It was designed to simplify shader programming, which was typically performed at the assembly language level. By interpretting textures and other graphics primitives as general purpose arrays, it is possible to implement general purpose programs. Regression and classification problems, using genetic programming, have already been successfully demonstrated (Chitty, 2007).

CUDA ("Compute Unified Device Architecture") is an NVIDIA hardware specific language for programming their more recent stream processors. Again, it is based on the C programming language and allows the user to write shader programs at a high level. CUDA also supports multiple GPU devices - and allows for different programs/data on each device. The toolkit provides fine grained control of how the GPU is utilised - potentially allowing for highly efficient use.

HLSL: Microsoft's High Level Shader Language (HLSL) is similar to Cg and CUDA. It is also a C-style language that can give some direct programmability of the shaders, but with the design tradeoff that they are used in conjunction with (rather than coded as part of) the source files in the higher level languages like C# and C++. With the current version of HLSL it is no longer necessary to use assembly-type instructions to access the shader functionality, and the language features C-like data, vector, and matrix types. Programs can be divided into C-like functions, and HLSL flow control includes both static and dynamic branching and looping. HLSL can be used to develop shader programs for DirectX that can work on both the Windows and XBox platforms. This has allowed Linear Genetic Programming to be implemented on the XBox and to benefit from the GPU (Wilson and Banzhaf, 2008).

It is interesting to note that Moore's Law is also valid for the growth of GPU performance over recent years, however with a different doubling parameter: While the doubling time for CPU performance is on the order of 18 months, doubling time for GPU performance is around 9-12 months. Thus, over time, the advantage of applications programmed for GPUs have over their CPU counterparts is bound to grow even faster.

The newest development on the hardware market are multiple GPU solutions. This is the next iteration of hardware improvement. For instance, the latest NVIDIA cards are essentially 2 of last years GPUs on one board. When coding for multiple devices, CUDA allows for completely different processes to run on each device.

5. Applications

At present most publications on using genetic programming with graphics hardware have been more concerned with proof of principle or demonstration of practical technology and algorithms rather than applying GPUs.

Large datasets are available in many different communities. For the demonstrations presented here, we chose one from the intrusion detection system community which was published for a competition in connection with the KDD conference 1999. We also discuss an image processing application, and a bioinformatics application. We finally demonstrate the feasibility of GPU computing on a gaming console.

The KDD 1999 IDS dataset

The full KDD Cup 1999 data set contains over four million entries. The data mining task was to find a network intrustion detector, that could discriminate between different types of network behaviour. The data itself has been critized, but we feel that the size of the dataset presents some challenges to conventional implementations. Because of memory restrictions on our current graphics card, we restrict ourselves to a subset of 10%, which contains 494,021 entries, each with 42 values. All input types were converted to floating point numbers (mapping string tokens to unique numbers). The dataset requires approximately 80 Mb of memory once in RAM.

Benchmark tests were performed using the RapidMind development platform, under Windows XP (Athlon 5200 CPU, 2 Gb RAM with a NVIDIA 8800 GTX, 768 Mb). We implemented Cartesian GP (CGP) (Miller and Thomson, 2000) for both the RapidMind GPU backend, and a straight C++ version. This allows us to easily compare timing for both a parallel and a conventional implementation. The CPU implementation uses only a single core. We chose not to use the RapidMind C++ backend as the overhead of invoking the native compiler is excessive - the system is better suited to situations where the parallel parts of the code only need to be compiled once.

For both the CPU and GPU, we used the function set of Table 15-1. We peformed evolutionary runs using both the GPU and CPU implementations, and recorded the number of Genetic Programming Operations Per Second (GPOps). Each run consited of evaluating 9005 individuals (200 generations, 50 indivdiduals with 5 elite individuals per population). We performed 10 runs of each implementation, and computed the timings based on statistics collected for each generation of each run.

For the GPU, peak performance was 694 million GPOps and an average of 388 million GPOps. In contrast, the CPU implementation peaked at 92 million GPOps with an average of 41 million GPOps.

On the GPU an individual took an average of 5.86 ms to evaluate, compared to 43.54 ms on the CPU – a speed up of 7.4 times on average. These timings include any overhead for data transfer and compilation of shader programs.

Image Filters

The evolution of image filters using Genetic Programming is a relatively unexplored task (but see (Poli, 1996)). This is most likely due to the high computational cost of evaluating the evolved programs. We have used a GPU implementation to tackle the challenge of reverse engineering image filters, i.e. to find the mapping between an image and the output of a filter applied to it. The filters we investigate in this paper are from the open source image processing program GIMP (GNU, 2008). To perform reverse engineering, again CGP is used to evolve programs acting as filters. These programs take a pixel and its neighbourhood from an image, and compute the next value of this central pixel. The convolution kernel is run on each pixel in an image producing a new image.

For much of the previous work on this problem a single, low resolution (256 x 256 pixel) image is used in the evaluation stage. This approach can be expected to result in over-fitting to a particular image. Thanks to the acceleration through the use of a GPU we were able to employ more images to help overcome such issues; here 16 different images (Figure 15-2) were used largely taken from the USC-SIPI image repository (12 used for fitness evaluation, and 4 for validation). This allows us to be confident that evolved filters will generalise well. As we are employing the GPU for acceleration, it is possible to test all images at the same time and obtain both the fitness and validation score simultaneously.

The original input images were combined together to form a larger image. A filter was applied using GIMP. We then employed the evolutionary algorithm to find the mapping between the input image and the output images. The fitness function attempts to minimize the error between the desired output (the GIMP processed image) and the output from the evolved filters.

Table 15-1 lists the available functions. A simple evolutionary algorithm with population of size 25, tournament selection (size 3) and elitism (best 3 individual) was used. We allowed evolution to run for 50,000 evaluations.

With this technique it was possible to evolve many different types of filters, such as *erode, dilate, emboss* and various edge detectors (see (Harding and Banzhaf, 2008) for more examples). Here we will illustrate the process with the Sobel filter, which is a type of directionless edge detector. Figure 15-3 shows how the evolved filter compares to the GIMP implementation.

Calculating the Genetic Programming Operations Per Second (GPOps) for our test system (NVIDIA 8800 GTX, AMD Athlon 3500+, Microsoft Accelerator API) a peak performance of 292 million GPOps and an average of 194 million GPOps was reached. Different operations appear to take different times

Table 15-1. CGP Function set used in the different applications. "x" means a particular instruction was used. The constants refer to the respective node's parameter.

Function	Description	KDD	Filters
ITERATION	Return the current iteration index		x
ADD	Add the two inputs	x	x
SUB	Subtract the second input from the first	x	x
MULT	Multiply the two inputs	x	x
DIV	Divide the first input by the second	x	x
ADD CONST	Add a constant to the first input	x	x
MULT CONST	Multiply the first input by a constant	x	x
SUB CONST	Subtract a constant from the first input	x	x
DIV CONST	Divide the first input by a constant	x	x
SQRT	Return the square root of the first input	x	x
POW	Raise first input to the power of second	x	x
COS	Return the cosine of the first input		x
SIN	Return the sin of the first input		x
NOP	No operation - return the first input		x
CONST	Return a constant		x
ABS	Return the absolute value of first input	x	x
MIN	Return the smaller of the two inputs		x
MAX	Return the larger of the two inputs		x
CEILING	Round up the first input	x	x
FLOOR	Round down the first input	x	x
FRAC	Return the fractional part of number		x
LOG2	Log (base 2) of the first input		x
RECIPRICAL	Return $1/firstinput$		x
RSQRT	Return $1/\sqrt{firstinput}$	x	x
CROOT	Return the cube root of first input	x	

Figure 15-2. The training and validation image set. All images are presented simultaneously to the GPU. The first column of images is used to compute the validation fitness, the remaining twelve for the training fitness. Each image is 256 by 256 pixels, with the entire image containing 1024 by 1024 pixels.

to execute. Across all the image filters we evolved, the peak performance was 324 Million GPOps, with an average of 145 GPOps. Using a CPU implemented reference driver, we were able to run the same experiments on the CPU. Execution yielded only 1.2 million GPOps, i.e. a factor of 100 times slower than the GPU. Considering the difference in performance, it would be impractical to run these experiments on the CPU with this size of image set.

A Bioinformatics Application

The applications from Bioinformatics are using the GPU primarily to speed up the fitness evaluation. In (Langdon and Harrison, 2008) GP was used to find a small set of GeneChip probes which indicated long term (10 or more years) survival of Breast Cancer.

From 1987 to 1989 during breast surgery tumour samples were saved and subsequently analysed using two GeneChips (HG-U133A and HG-U133B). These two chips together collect more than a million expression values for 251 patients (Miller, 2005). A major part of the data mining exercise was simply to use evolution to decide which of these are useful at predicting the patients future. GP was run multiple times in a series of passes to winnow the useful data

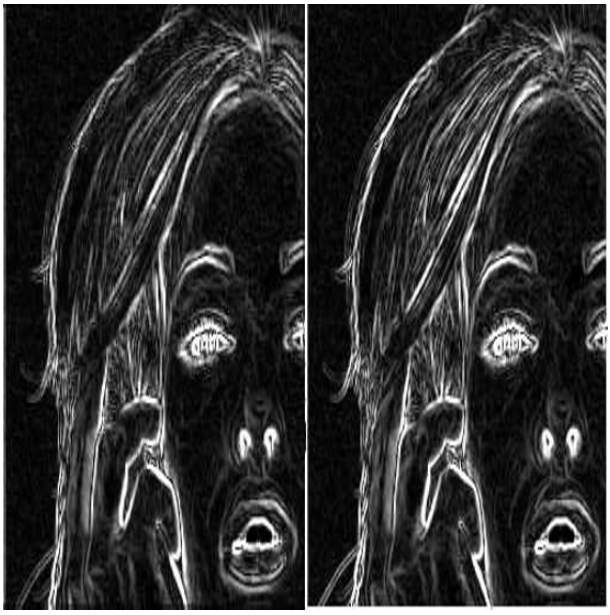


Figure 15-3. Evolving a Sobel edge detector. The left image shows the output from the evolved filter. The right image is the output from the GIMP filter. The two outputs are visually very similar.

from the remaining chaff (Langdon and Buxton, 2004). This involved running the GP system with a population of five million programs hundreds of times on hundreds of megabytes of training data. The final predictor was a non-linear function of only a handful of the million initial variables.

A second study (Langdon, 2008) also involved the use of GeneChips. However, GP was used here to locate problems with Affymetrix technology itself rather than looking at the medical results of applying that technology. Affymetrix Chips provide many (typically 22) separate readings for each human gene. If the GeneChip was well designed, all the readings should be well correlated. However our measurements of the correlation found thousands of cases where this was not true (Upton et al., 2008). GP was used to find patterns in the data to indicate which readings were unreliable. This has led to discussion about the underlying physics of the GeneChip, what may cause the problems and how they might be avoided in future designs (Upton et al., 2008).

XBOX Application

We have used the GPU processing power of a video game console, namely of the Xbox 360 (Andrews and Baker, 2006) for general computation. In late 2006, Microsoft launched XNA Game Studio Express, which integrated with

C# Studio Express, and thus became the first video game console to provide its users with access to its GPU. By using XNA with HLSL, we designed GPGPU applications and ported them to an Xbox 360 for execution. The application itself was a Linear GP program, which implemented both fitness evaluation and mutation using the GPU. Production with the XNA framework targeted to the Xbox 360 presented interesting design challenges.

An XNA project interface mandates that initialization (Initialize), update of program logic (Update), and rendering of graphics (Draw) methods be implemented. The program runs by repeatedly updating the Update and Draw methods — it is designed to be a video game that is constantly checking its logic status and updating the graphics on the screen. The Draw method is thus the main component of our GPGPU implementation, as this is where the shader programs on the GPU are called from. Rather than use a typical loop construct for LGP tournament execution, the repeated execution of the Draw method is harnessed to conduct generational tournaments over numerous trials.

The Draw method, as appropriate, would process the textures using HLSL programs loaded at compile-time or otherwise conduct CPU-side GP tournament processes. The population of LGP individuals is represented as multiple textures. In particular, two textures represent the content of the instructions, and another texture represents the registers. Multiple passes are used to examine the instruction in the first two textures and place subresults in the third register texture. A single large texture represents the fitness cases. Mutation was implemented with a texture of randomly chosen probabilities and a corresponding texture of potential chromosome replacements. The work (Wilson and Banzhaf, 2008) established, for the first time, how to use a video game console GPU for general computation in any capacity, and how to use a console in general (both CPU and GPU) for genetic programming. GPU implementations on the XBox 360 were found to be moderately faster than their CPU equivalents, but this is likely indicative of the tightly integrated nature of the CPU and GPU in the Xbox 360 architecture. Practically speaking, the results demonstrate that future GPGPU programmers of the Xbox 360 need not be as focused on performance consequences of placing functionality with the GPU as opposed to the CPU.

6. Perspective

In this chapter we have motivated the use of GPUs for Genetic Programming through the excessive computational demands GP poses on every system. We have explained the general approach to using a GPU, given an overview of currently usable software systems, and demonstrated by way of example a number of interesting applications of GP on GPUs. Our work succeeded in porting various aspects of linear, tree and graph GP systems onto GPU platforms.

It is too early to predict which types of parallelization of GP using GPUs will in the end be the most effective. For example, it would be helpful to identify categories of fitness functions that may and may not be suitable for implementation on GPUs. It appears that at first glance, many GP problems are easily mappable onto the GPU hardware. Since it can be expected that in the future the APIs will hide even more functionality implementations should become easier to acheive.

We hope that this chapter motivates many more researchers to turn their attention to this new platform that promises to revolutionize how Genetic Programming is performed.

Acknowledgment

We would like to thank Nolan White for helping us get the hardware working, and RapidMind for providing us with their code free of charge. WB further acknowledges support by NVIDIA.

References

Andre, David and Koza, John R. (1996). A parallel implementation of genetic programming that achieves super-linear performance. In Arabnia, Hamid R., editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume III, pages 1163–1174, Sunnyvale. CSREA.

Andrews, J. and Baker, N. (2006). Xbox 360 system architecture. *IEEE Micro*, 26:25–37.

Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998). *Genetic Programming - An Introduction*. Morgan Kaufmann, San Francisco, CA, USA.

Banzhaf, Wolfgang and Lasarczyk, Christian W. G. (2004). Genetic programming of an algorithmic chemistry. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 11, pages 175–190. Springer, Ann Arbor.

Bennett III, Forrest H, Koza, John R., Shipman, James, and Stiffelman, Oscar (1999). Building a parallel computer system for $18,000 that performs a half peta-flop per day. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1484–1490, Orlando, Florida, USA. Morgan Kaufmann.

Chitty, Darren M. (2007). A data parallel approach to genetic programming using programmable graphics hardware. In *GECCO 07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1566–1573, New York, NY, USA. ACM.

Chong, Fuey Sian and Langdon, W. B. (1999). Java based distributed genetic programming on the internet. In Banzhaf, Wolfgang, Daida, Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1229, Orlando, Florida, USA. Morgan Kaufmann. Full text in technical report CSRP-99-7.

Eklund, Sven E (2003). Time series forecasting using massively parallel genetic programming. In *Proceedings of Parallel and Distributed Processing International Symposium*, pages 143–147.

Feng, W., Warren, M., and Weigle, E. (2002). The bladed beowulf: A cost-effective alternative to traditional beowulfs. In *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, page 245, Washington, DC, USA. IEEE Computer Society.

Fernández, F., Tomassini, M., and Vanneschi, L. (2003). An Empirical Study of Multipopulation Genetic Programming. *Genetic Programming and Evolvable Machines*, 4(1):21–51.

Folino, G., Forestiero, A., and Spezzano, G. (2006). A Jxta based asynchronous Peer-to-Peer Implementation of Genetic Programming. *JOURNAL OF SOFTWARE*, 1(2):13.

Folino, G., Pizzuti, C., and Spezzano, G. (2003a). A scalable cellular implementation of parallel genetic programming. *Evolutionary Computation, IEEE Transactions on*, 7(1):37–53.

Folino, G., Pizzuti, C., and Spezzano, G. (2003b). Ensemble Techniques for Parallel Genetic Programming Based Classifiers. *Genetic Programming: 6th European Conference, EuroGP 2003, Essex, UK, April 14-16, 2003: Proceedings*.

GNU (2008). Gnu image manipulation program (GIMP). www.gimp.org. [Online; accessed 21-January-2008].

Gross, R., Albrecht, K., Kantschik, W., and Banzhaf, W. (2002). Evolving chess playing programs. In Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., and Jonoska, N., editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 740–747, New York. Morgan Kaufmann Publishers.

Harding, S. and Banzhaf, W. (2007a). Fast genetic programming on GPUs. In Ebner, M., O'Neill, M., Ekart, A., Vanneschi, L., and Esparcia-Alcazar, A., editors, *Proc. 10th Europ. Conference on Genetic Programming, Valencia, Spain*, volume 4445 of *LNCS*, pages 90 – 101. Springer.

Harding, S. and Banzhaf, W. (forthcoming, 2008). Image filters evolved with genetic programming. Technical Report, Department of Computer Science, Memorial University of Newfoundland.

Harding, Simon (2008). Evolution of image filters on graphics processor units using cartesian genetic programming. In *WCCI*. IEEE.

Harding, Simon and Banzhaf, Wolfgang (2007b). Fast genetic programming and artificial developmental systems on GPUs. In *Proceedings of the Symposium on High Performance Computing Systems (HPCS-2007)*. IEEE, DOI: http://doi.ieeecomputersociety.org/10.1109/HPCS.2007.17.

Harris, M. (2005). Mapping computational concepts to gpus. In Pharr, M. and Fernando, R., editors, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General Purpose Computation*, pages 493–508. Addison-Wesley, Boston, MA.

Hillis, W.D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234.

INTEL (accessed March 20, 2008). Tom's hardware's 2007 cpu charts.

Juille, H. and Pollack, J.B. (1995). Parallel genetic programming and fine-grained SIMD architecture. *Working Notes for the AAAI Symposium on Genetic Programming*, pages 31–37.

Koza, John R., Bennett III, Forrest H, Hutchings, Jeffrey L., Bade, Stephen L., Keane, Martin A., and Andre, David (1997). Rapidly reconfigurable field-programmable gate arrays for accelerating fitness evaluation in genetic programming. In Koza, John R., editor, *Late Breaking Papers at the 1997 Genetic Programming Conference*, pages 121–131, Stanford University, CA, USA. Stanford Bookstore.

Langdon, W. B. (2004). Global distributed evolution of L-systems fractals. In Keijzer, Maarten, O'Reilly, Una-May, Lucas, Simon M., Costa, Ernesto, and Soule, Terence, editors, *Genetic Programming, Proceedings of EuroGP'2004*, volume 3003 of *LNCS*, pages 349–358, Coimbra, Portugal. Springer-Verlag.

Langdon, W. B. (2008). Evolving GeneChip correlation predictors on parallel graphics hardware. In *Proceedings of the IEEE World Congress on Computational Intelligence*, Hong Kong. IEEE. Forthcoming.

Langdon, W. B. and Buxton, B. F. (2004). Genetic programming for mining DNA chip data from cancer patients. *Genetic Programming and Evolvable Machines*, 5(3):251–257.

Langdon, W. B. and Harrison, A. P. (2008). GP on SPMD parallel graphics hardware for mega bioinformatics data mining. *Soft Computing*. Special Issue. On line first.

Langdon, W. B. and Poli, Riccardo (2002). *Foundations of Genetic Programming*. Springer-Verlag.

Langdon, W.B. and Banzhaf, W. (2008). A SIMD interpreter for genetic programming on GPU graphics cards. In O'Neill, M., Vanneschi, L., Esparcia-Alcazar, A., and Ebner, M., editors, *Proc. 11th Europ. Conference on Genetic Programming, Valencia, Spain*, volume 4971 of *LNCS*, pages 70 – 79. Springer.

LibSh Wiki (accessed March 11, 2008). Libsh sample code. `http://www.libsh.org/wiki/index.php/Sample_Code`.

Mark, W., Glanville, S., and Akeley, K. (2003). Cg: A system for programming graphics hardware in a c-like language. *ACM Transactions on Graphics*.

Martin, Peter (2001). A hardware implementation of a genetic programming system using FPGAs and Handel-C. *Genetic Programming and Evolvable Machines*, 2(4):317–343.

Miller, J.F. and Thomson, P. (2000). Cartesian genetic programming. *Genetic Programming, Proceedings of EuroGP*, 1802:121–132.

Miller, L. et al. (2005). An expression signature for p53 status in human breast cancer predicts mutation status, transcriptional effects, and patient survival. *PNAS*, 102:13350 – 13355.

NVIDIA (accessed March 11, 2008). http://developer.nvidia.com/page/cg.

NVIDIA (November 2006). NVIDIA Technical Brief: NVIDIA GeForce 8800 GPU Architecture Overview.

Oussaidene, M., Chopard, B., Pictet, O.V., and Tomassini, M. (1996). Parallel genetic programming: An application to trading models evolution. *Genetic Programming*, pages 357–380.

Page, J., Poli, R., and Langdon, W. B. (1999). Smooth uniform crossover with smooth point mutation in genetic programming: A preliminary study. In Poli, Riccardo, Nordin, Peter, Langdon, William B., and Fogarty, Terence C., editors, *Genetic Programming, Proceedings of EuroGP'99*, volume 1598 of *LNCS*, pages 39–49, Goteborg, Sweden. Springer-Verlag.

Poli, Riccardo (1996). Genetic programming for image analysis. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 363–368, Stanford University, CA, USA. MIT Press.

RapidMind (accessed March 11, 2008). Libsh. `http://libsh.org/`.

Robertson, G.G. and Riolo, R.L. (1988). A tale of two classifier systems. *Machine Learning*, 3:139–159.

Stoffel, Kilian and Spector, Lee (1996). High-performance, parallel, stack-based genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 224–229, Stanford University, CA, USA. MIT Press.

Tarditi, David, Puri, Sidd, and Oglesby, Jose (2006). Msr-tr-2005-184 accelerator: Using data parallelism to program GPUs for general-purpose uses. Technical report, Microsoft Research.

Turton, Openshaw, and Diplock (1996). Some geographic applications of genetic programming on the Cray T3D supercomputer. In Jesshope, Chris R. and Shafarenko, Alex V., editors, *UK Parallel'96*, pages 135–150, University of Surrey. Springer.

Upton, G.J.G., Langdon, W.B., and Harrison, A.P. (in preparation, 2008). GGGG probes in microarrays are not gene-specific.

Wikipedia (accessed March 11, 2008). http://www.wikipedia.org/wiki/flops.

Wilson, G. and Banzhaf, W. (2008). Linear Genetic Programming GPGPU on Microsoft's XBox 360. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2008), June 1-6, 2008, Hong Kong, China*.

XBOX (accessed March 20, 2008). Xbox 360 technical specifications.

Ziegler, Jens and Banzhaf, Wolfgang (2003). Decreasing the number of evaluations in evolutionary algorithms by using a meta-model of the fitness function. In Ryan, Conor, Soule, Terence, Keijzer, Maarten, Tsang, Edward, Poli, Riccardo, and Costa, Ernesto, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 264–275, Essex. Springer-Verlag.

Chapter 16

GENETIC PROGRAMMING FOR INCENTIVE-BASED DESIGN WITHIN A CULTURAL ALGORITHMS FRAMEWORK

Mostafa Z. Ali[1], Robert G. Reynolds[2] and Xiangdong Che[2]

[1]*Department of Computer Science, Jordan University of Science and Technology, Irbed, Jordan*
[2]*Department of Computer Science, Wayne State University, Detroit, Michigan*

Abstract Cultural Algorithms employ a basic set of knowledge sources, each related to knowledge observed in various social species. Here, a single influence function that is designed to integrate together the diverse knowledge sources in order to coordinate their problem solving efforts in an optimization problem. The influence function currently has two components. First, the default influence function in the Cultural Algorithms Toolkit is based upon the Marginal Value Theorem based upon basic predator-prey relationships from Population Biology. This controls the portion of the space that each knowledge source will be investigating. Second, the Social Fabric component of the influence function determines the individuals that the knowledge sources will be influencing in the social network. The Social Fabric component has several parameters that can be used to design its structure for a given problem. Different parameter combination can lead to different system performance for a given problem as will be demonstrated later in the paper. If our cultural system needs to solve several different problems over time, then a parameter combination and derived Social Fabric that is optimal for one problem may not be optimal for another. Our goal is to use Genetic Programming to evolve a Decision Tree that selects the appropriate parameter combination for a given problem situation. An example of the approach is given for a class of Engineering Optimization Problems, the Speed Reducer.

Keywords: cultural algorithms, decision trees, incentive-based design, social fabric, social networks, agent-based models, engineering design optimization, speed reducer problem.

Cultural Algorithms can provide a flexible framework in which to embed the Genetic Programming (GP) process. In particular, we are interested in using Cultural Algorithms to support the social aspects of programming and

problem solving (Reynolds, 1994; Reynolds and Peng, 2004). On the one hand, we demonstrated a prototype Cultural Algorithm that coordinated a team of GP programming specialists by collecting information about the progress of various aspects of the design for use by individual GPs on selected subtasks (North et al., 2006). On the other hand, we used Genetic Programming as a vehicle to modify the design of an incentive-based rebate plan to fit changing purchasing strategies in an agent-based customer model.[1]

In this paper we continue our investigations regarding a GP approach to incentive-based design within social networks using a Cultural Algorithm framework. The motivational example is as follows. A number of competing knowledge sources wish to control and influence individuals within a social network by providing a subset of individuals certain incentives. Thus, each knowledge source wishes to control a portion of the network. If they are successful they can expand their control over time. This is a classic problem in incentive based design. In order to accomplish this we have embedded the Cultural Algorithms framework within the recursive porous agent simulation tool (Repast), producing a toolkit that called Cultural Algorithms Simulation Toolkit (CAT)(North et al., 2006).

Here, there is a single influence function is designed to integrate together the diverse knowledge sources in order to coordinate their problem solving efforts in an optimization problem. The influence function currently has two components. First, the default influence function in the Cultural Algorithms Toolkit is the Marginal Value Theorem based upon basic predator-prey relationships from Population Biology. This controls the portion of the space that each knowledge source will be investigating. Second, the Social Fabric component of the influence function determines the individuals that the knowledge sources will be influencing in the social network. Each knowledge source attempts to influence a subset of individuals connected within a social network. Each individual then distributes that potential influence to those individuals directly linked to it in the network. As a result, an individual may have more than one competing request based on its position in the network, one that it gets directly from a knowledge source and others that it gets indirectly from its neighbors as a result of its connectedness in the network. If there are competing requests, then there needs to be a mechanism to select the winner who will control the individual. The default mechanism here is majority rule. If the mechanism does not produce a single knowledge source then a conflict resolution mechanism is used.

The Social Fabric component therefore has several parameters that can be used to design its structure for a given problem. Different parameter combinations can lead to different system performance for a given problem as will be demonstrated later in the paper. If our cultural system needs to solve several

[1] http://repast.sourceforge.net

different problems over time, then a parameter combination and derived Social Fabric that is optimal for one problem may not be optimal for another. Our goal is to use Genetic Programming to evolve a Decision Tree that selects the appropriate parameter combination for a given problem situation. So, the terminal set for the evolved Decision Tree will be a parameter combination for the Social Fabric component of the Influence Function. The function set for the tree will be Conditional and Boolean functions that operate on a set of problem parameters to decide the most effective set of parameter combinations at the leaves. By doing this we will be able to see how the social organization or fabric in a Cultural System is a response to the complexity and variety of problems posed to it. Since the process of controlling the social network is viewed metaphorically as "weaving a social fabric", we are seeking to use Genetic Programming to develop a controller for the Social Fabric component that weaves a social fabric together to allow the system to solve different problems.

Our goal then is to examine the potential feasibility of this approach to design optimization using Genetic Programming to evolve the parameters for the Social Fabric component of the CAT influence function. In this paper, we begin by describing the Cultural Algorithm framework in Section 1. In Section 2 we introduce the Social Fabric influence function component and describe the parameters that are currently used to weave a fabric for a given problem. We then apply our approach to the optimal design of a Speed Reducer component in Section 3. Section 4 gives our conclusions. In future work we anticipate adding a third component to the influence function that uses n-player games to resolve conflict when the selected mechanism, majority vote in this case, fails to identify a unique candidate knowledge source. The Genetic Programming framework could also be used in designing that component as well.

1. The Cultural Algorithms Configuration

Cultural Algorithms

The basic pseudo code of Cultural Algorithms (CA) is shown in Equation 1.0 where P^t represents the Population component at time t, and B^t for the Belief Space at time t. The algorithm begins by initializing both the Population and the Belief Space. Then it enters the evolution loop until the termination condition is satisfied.

At the end of each generation, individuals in the Population Space are first evaluated using a performance function, *obj()*. An acceptance function, *accept()* is then used to determine which individuals will be allowed to update the Belief Space. Experiences of those chosen elite individuals are then added to the contents of the Belief Space via function *update()*. This update activity can utilize various forms of symbolic learning activity. Next, knowledge from the belief space is allowed to influence the selection of individuals for the next

generation of the population through the *influence()* function. The two feedback paths of information, one through the *accept()* and *influence()* functions, and the other through individual experience and the *obj()* function create a system of dual inheritance in both the population and the belief space.

The CA repeats this process for each generation until the pre-specified termination condition is met. In this way, the population component and the belief space interact with and support each other, in a manner analogous to the evolution of human culture.

```
Begin
    t = 0;
    initialize Bt, Pt
    repeat
        evaluate Pt {obj()}
        update(Bt, accept(Pt))
        generate(Pt, influence(Bt))
        t = t + 1;
        select Pt from Pt - 1
    until (termination condition achieved)
end
```

The Cultural Algorithm Knowledge Sources

Five basic categories of cultural knowledge have been identified: Normative Knowledge, Situational Knowledge, Domain Knowledge, History Knowledge, and Topographical Knowledge. In our approach each knowledge source vies to influence a subset of the social networks associated with individuals in the population space. Normative Knowledge is a set of promising variable ranges that provide standards for individual behaviors and guidelines within which individual adjustments can be made (Chung and Reynolds, 1998). Normative Knowledge leads individuals to "jump into the good range" if they are not already there.

Situational Knowledge provides a set of exemplary cases that are useful for the interpretation of specific individual experience. Situational Knowledge leads individuals to "move toward the exemplars."

Topographical Knowledge was originally proposed to reason about region-based functional landscape patterns (Jin and Reynolds, 1999). The whole landscape is divided into cells according to spatial characteristics and each cell keeps track of the best individual in its region. Topographical Knowledge guides individuals to emulate the cell-best (similar to local optima).

Domain Knowledge uses knowledge about the problem domain to guide search. For example, in a functional landscape composed of cones, knowledge

about cone shape and the related parameters will be useful in reasoning about them during the search process.

Historical or Temporal Knowledge monitors the search process and records important events in the search. These important events may be either a significant change in the search space or a detection of landscape change. Individuals guided by the History Knowledge can consult those recorded events for guidance in selecting a move direction.

2. Weaving the Social Fabric Influence Function Concept

The default mechanism for the CAT influence function is the based upon the Marginal Value Theorem from population biology in which the Knowledge Sources are viewed metaphorically to be the "predators" and the performance environment is the resource or "prey" to be exploited. It assumes that each knowledge source generates individuals within a "patch" or region within the solution space in order to probe or sample that space. A knowledge source (predator) will move its patch when its performance drops below an expected level of performance. This approach will theoretically maximize the resource intake of the each knowledge source over time. This component of the influence function derives from the social behavior of animal populations. It has been shown that this approach performs better than the just random association of knowledge sources with points in the solution space.

From a theoretical perspective, we view individuals in a human social system as participating in a variety of different networks. Several layers of such networks can be supported within a population. The interplay of these various network computations is designated as the "social fabric." This notion of social fabric has appeared metaphorically in various ways within Computer Science. For example, IBM among others developed tools to reinforce the "social fabric" whereby designers and programmers interact to solve complex problems (Cheng et al., 2005).

The social fabric is viewed as a computational tool that influences the action and interaction of the various knowledge sources. We can add this as a second component to the influence function to represent the impact that a social network can have on the solution process. Whereas, the Marginal Value Theorem addresses where the knowledge sources are generating individual probes in the problem space, the Social Fabric influence component addresses who the individuals are in the network that the knowledge sources will influence. Adding the Social Fabric component to the Marginal Value Theorem gives us an indication of how the addition of an explicit culturally based social system can influence problem solving performance over that for animal populations in general.

Informally, we have N networks and M individuals. An individual can be associated with one or more networks. For a given network only certain information is allowed to flow along that network between nodes. Each network can be viewed as being produced by a single thread that links up the participating nodes. Information, matter, and energy can flow along the thread. There is a set of connector threads that are used to weave the threads together to produce the social fabric. They do that by associating a node in one network with a node in another such that flow in one network can constrain flows in the other and vice versa. The tightness of the weave is reflected in the number of connections between the networks. If the weave is too tight then there is less flexibility when the system is placed under stress.

In Figure 16-1, the concept is illustrated schematically with 5 different networks given as grey-shaded vertical lines. Individuals are given as horizontal lines with a node representing a possible participation in each network. The node is blank if the individual does not participate in that particular network. It is darkened if it participates sometimes, and darkened and circled if it is a frequent participant.

The individuals are ordered from highest participation to lowest participation of the five networks. The topologies of the individual networks are not shown here, just the extent to which the networks are woven together by the participation of the individuals in the knowledge sources activities. Notice that in this example only one individual is active in all networks. If that individual was removed, the "weave" will fall apart since it binds all of the networks together through its participation. Notice also that some groups such as the first vertical group of circles from left, are characterized by a small but active set of participants. On the other hand, the fourth group is an example of one where everyone participates somewhat. For those individuals who participate in more than one group, activities in one group can constrain activities in another.

Weaving the Social Fabric into the CAT System

As a simple configuration in CAT we can simply specify just one network, one that is accessible to the Knowledge Sources in the Belief Space. What we wish to investigate is whether just having access to the Social Fabric is sufficient for the Knowledge Sources to improve the performance of the influence function as opposed to not having a network to distribute their influence over as with the Marginal Value component by itself. Each configuration of the Social Fabric in CAT is specified by a set of internal parameters. We now motivate and describe each.

The process starts where each individual will be affected by one knowledge source (as a special case) that will represent the initial signal to be passed to other individuals. The signal is passed to adjacent individuals in the network based

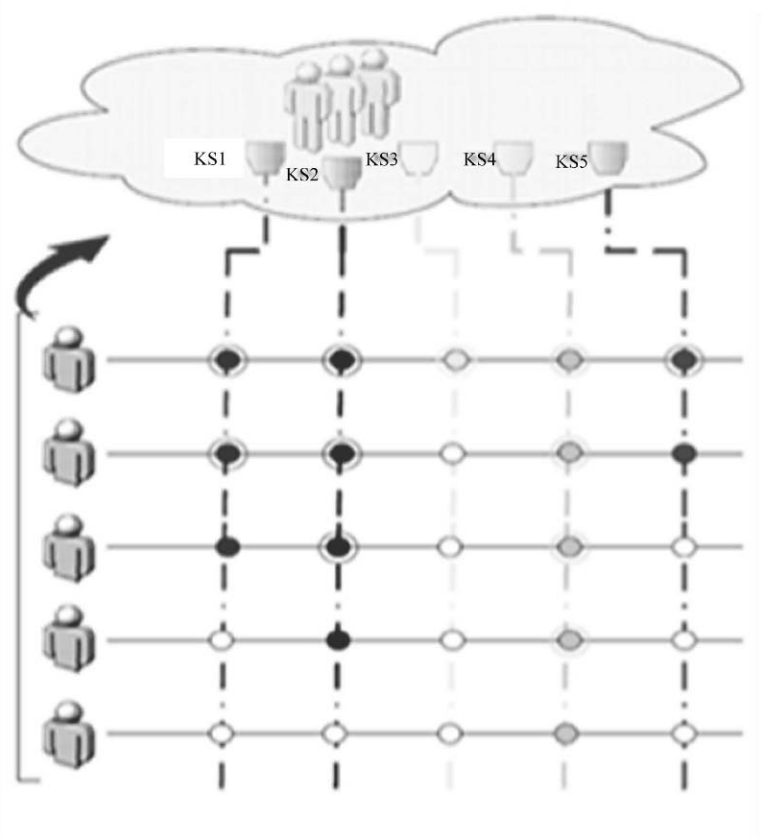


Figure 16-1. An example of the Social Fabric. The Fabrics run longitudinally while the participants who will do the "weaving" are on the horizontal axis.

on the network connectivity. The individual is represented as a node, where the number of connections or hops over which it can transmit this information to its neighbors corresponds to its influence. The maximum number of hops can be either 0 or d meaning either no connections or d connections at a time. In order to start with the simplest case, we will assume that each individual is connected to a fixed number of other individuals using a constant topology. The network topologies that we used here were taken from work in Particle Swarm Optimization where the impact of various topologies on the communication of local information among particles has been studied.

Several frequently used topologies taken from the Particle Swarm Optimization literature are supported in CAT. For example, the *lbest* model is the simplest form of a local topology known as the ring model. The lbest ring model connects each individual to only two other individuals in the population and is shown in Figure 16-2 left. Another frequently used topology is the *gbest* topology. In

this topology each individual in the network is connected to all the individuals in the network as shown in Figure 16-2 right. One advantage of the lbest model may lie in its slower convergence rate relative to the gbest model which may reduce the chance of premature convergence to a false peak.

Another topology supported in CAT is the square topology in which each individual has four connections to other individuals in the population. The square topology represents the notion of a "small world network." Small World networks are characterized by having 4-7 strong links between individuals. This type of network is commonly seen in kinship based rural societies.

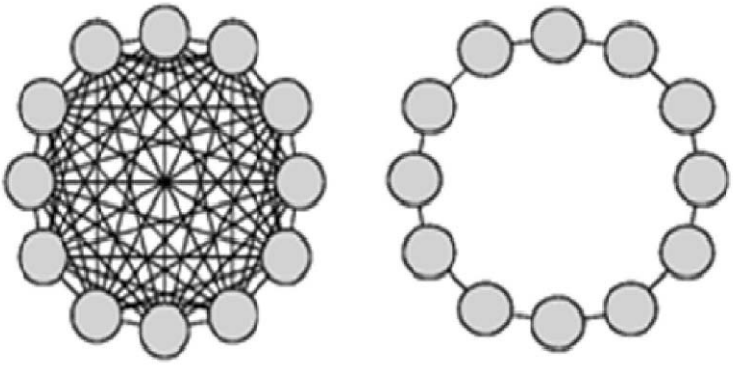


Figure 16-2. Topologies used in the Social Fabric model for connection between individuals. (left) lbest ring topology. (right) gbest topology.

In this simple version of Knowledge Source interaction, Knowledge Sources do not know anything about the network and the selected individuals' position in it. The individual then transmits the name of the influencing Knowledge Source to its neighbors through as many hops as specified by the configuration parameter. Next, each node counts up the number of Knowledge Source bids that it collects. It will have the direct influence from the Knowledge Source that selected it plus the names of the Knowledge Sources transmitted to it by its neighbors. The Knowledge Source that has the most votes is the winner and will direct the individual for that time step. In case of a tie, there are several simple mechanisms currently implemented in CAT. They include, select the "most frequently used Knowledge Source," "the least frequently used Knowledge Source," and "the Knowledge Source that directly influenced the individual," among others.

The goal here was to use Genetic Programming to evolve a Decision Tree that selects an appropriate Social Fabric configuration based upon information about external problem parameters. Each configuration is a potential terminal node of the Decision Tree. It is currently a four-tuple that consists of the following parameters: (NHOPS (number of hops), TOP (network topology), MECH (tie-

break mechanism), and FREQ (frequency with which influences are distributed in the network)).

The range for NHOPS is from 0 to N where N is the number of individuals or nodes in the network. The range for TOP is $lbest, gbest, square$. The range for FREQ is $1, ..., MAX\_GEN$ where MAX_GEN is the maximum number of allowed generations. The range for the MECH variable is $LFREQ, MFREQ, DIRECT, RANDOM$. So, a possible configuration is (1, square, 4, Direct).

The body of the tree consists of a Function set defined over $IFTHENELSE, and, or, not, gt., lt., =$. The Function set is defined relative to a set of external problem parameters. The idea is that the Genetic Programming system evolves a Decision Tree that is employed for use by the CAT system over a set of different problems, where each problem is given as a set of external problem variables. For example, along with a set of Engineering problems in the CAT system there is a "Cones World" where resource cones are distributed over a problem landscape and the goal is to locate the cone with the optimal peak. A problem class in the Cones World can be described as a tuple in terms of the Number of Cones to be placed on the landscape (N), and the complexity coefficient, alpha. The latter specifies a distribution of the cones that can be of low complexity to chaotic in terms of a set of more detailed parameters such as cone height, base etc. An example tree is Lisp-notation for a Cones World problem is as follows:

(IF (>, n, 100) THEN (1, square, 4, direct) ELSE (1, lbest, 2, RANDOM).

In the remainder of the paper we investigate the nature of the Social Fabric configuration for one of the Engineering problems included in CAT, the Speed Reducer problem. As a proof of principle, only one problem configuration will be used so the tree will have only one level. In future work we will look to combining multiple problem environments to produce a more complex tree.

3. The Design of A Speed Reducer

Golinski's speed reducer problem (Golinski, 1970; Ray, 2003) is one of the most well-known problems of the NASA Langley Multidisciplinary Design Optimization (MDO) Test suite. This problem represents the design of a simple gearbox such as might be used in a light airplane between the engine and propeller to allow each to rotate at its most efficient speed. The gearbox is depicted in Figure 4 and its seven design variables are labeled.

Golinski modeled the speed reducer so as to minimize its weight subject to constraints on bending stress of the gear teeth, traverse deflections of the shafts, and stresses in the shaft. Seven variables are used for describing the problem here. They are:

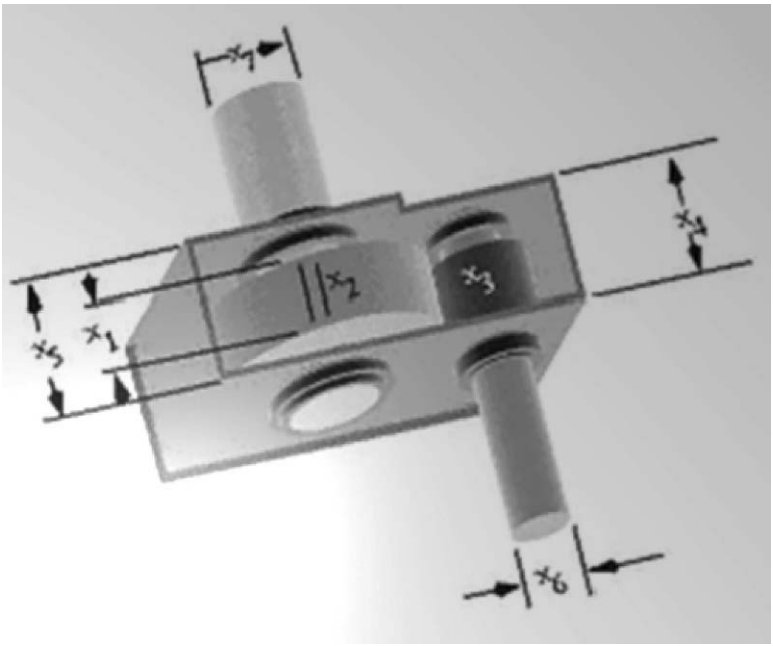


Figure 16-3. The Gearbox used in Golinski's Speed Reducer Problem.

x1: face width x2: module of teeth x3: number of teeth in the pinion x4: length of the first shaft between bearings x5: length of the second shaft between bearings x6: diameter of the first shaft x7: diameter of the second shaft The third variable is integer, while the rest of the variables are continuous.

The Minimization of the weight of a speed reducer is the most complex problem among the benchmark problems we have available in the CAT system in terms of number of dimensions, problem constraints, domain constraints, and complexity of the environment surface. The configured parameters for the best generated configuration are shown in Table 16-1.

Table 16-1. Parameters used for Problem P6 using Square Topology.

Parameter	Value
# of Individuals	100
# of Generations	1000
Fitness Algorithm	P6
Topology Used	*SQUARE*
Conflict Resolution	Direct
Frequency	3

The related tree is given as:

(IF (for the variable ranges given below) THEN (1, square, 3, direct))

The goal is to minimize:

$$f(\overrightarrow{x}) = 0.7854x_1x_2^2(3.3333x_3^2+14.9334x_3-43.0934)-1.508x_1(x_6^2+x_7^2)+$$

$$7.477(x_6^3+x_7^3)+0.7854(x_4x_6^2+x_5x_7^2)$$

Subject to:

$$g_1(\overrightarrow{x}) = \frac{27}{x_1x_2^2x_3} - 1 \leq 0$$

$$g_2(\overrightarrow{x}) = \frac{397.5}{x_1x_2^2x_3^2} - 1 \leq 0$$

$$g_3(\overrightarrow{x}) = \frac{1.93x_4^3}{x_2x3x_7^4} - 1 \leq 0$$

$$g_4(\overrightarrow{x}) = \frac{((\frac{745x_4}{x_2x_3})^2+16.9\times10^6)^{\frac{1}{2}}}{110.0x_6^3} - 1 \leq 0$$

$$g_5(\overrightarrow{x}) = \frac{((\frac{745x_5}{x_2x_3})^2+157.5\times10^6)^{\frac{1}{2}}}{85.0x_7^3} - 1 \leq 0$$

$$g_6(\overrightarrow{x}) = \frac{x_2x_5}{40} - 1 \leq 0$$

$$g_7(\overrightarrow{x}) = \frac{5x_2}{x_1} - 1 \leq 0$$

$$g_8(\overrightarrow{x}) = \frac{x_1}{12x_2} - 1 \leq 0$$

$$g_9(\overrightarrow{x}) = \frac{1.5x_6+1.9}{x_4} - 1 \leq 0$$

$$g_{10}(\overrightarrow{x}) = \frac{1.1x_7+1.9}{x_5} - 1 \leq 0$$

Where:

$$2.6 \leq x_1 \leq 3.6, 0.7 \leq x_2 \leq 0.8, 17 \leq x_3 \leq 28,$$

$$7.3 \leq x_4 \leq 8.3, 7.8 \leq x_5 \leq 8.3, 2.9 \leq x_6 \leq 3.9, 5.0 \leq x_7 \leq 5.5.$$

The performance of the best configuration is described below in Figure 16-4 to Figure 16-11. Figure 16-4 through Figure 16-7 show the population swarms produced by each of the controlling knowledge sources in terms of two of the variables, module ($x2$) and number of teeth ($x3$). Each individual's position is encoded by a geometrical symbol that relates to the Knowledge Source that directs it. In dimensions (x2, x3), Topographic knowledge produces a result near the optimum during the coarse-grained search phase. This attracts the Situational, Domain, Normative, and History knowledge sources in the fine-grained phase, as shown in Figure 16-4 through Figure 16-7.

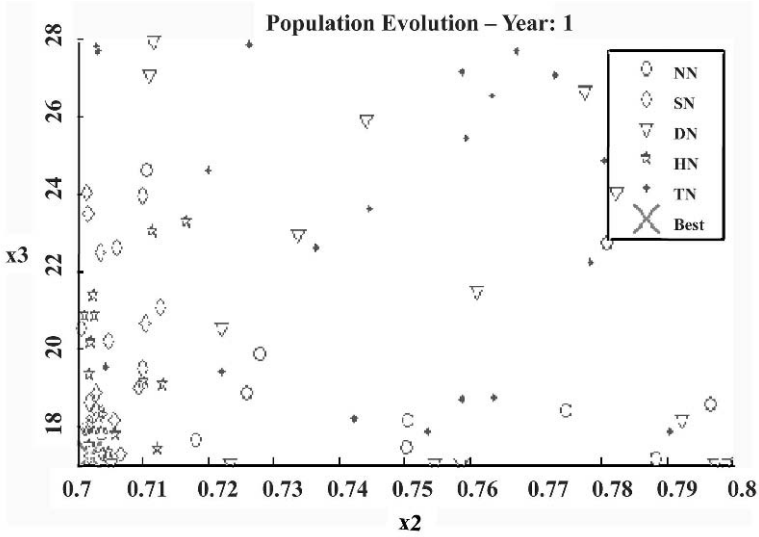


Figure 16-4. Population Swarm Plot of Dimension (x2, x3) at Generation 1.

Figure 16-8 through Figure 16-11 give the meta-level bounding boxes that describe the distribution of individuals generated by each Knowledge Source from generation 1 through 8 when the optimum has been found. Each bounding box is given by the mean and standard deviation of the individuals generated and coded with the Knowledge Source name. The bounding boxes for Topographic and Domain knowledge sources encompass most of the problem space in this dimension in Figure 16-8. Notice that by generation 5, the bounding box for Situational knowledge has focused the search around the optimal value and is channeling new individuals into that area primarily causing the other Knowledge Sources to "swarm" towards it. By generation 8, the bounding boxes have very small areas which show how focused and specialized they are.

Other configurations were manually generated and tested for the problem. The manual configuration of the parameters represents modifying the variables "Frequency," "Configuration_Topology," "Conflict_Resolution," and keeping

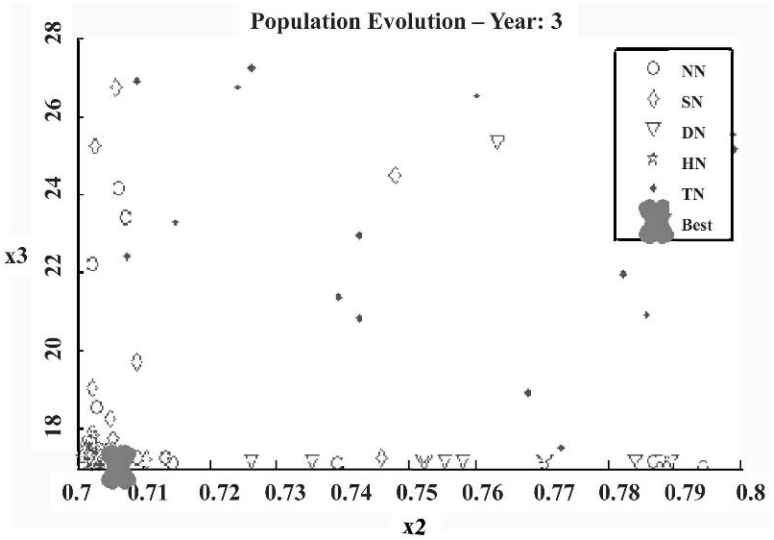


Figure 16-5. Population Swarm Plot of Dimension (x2, x3) at Generation 3.

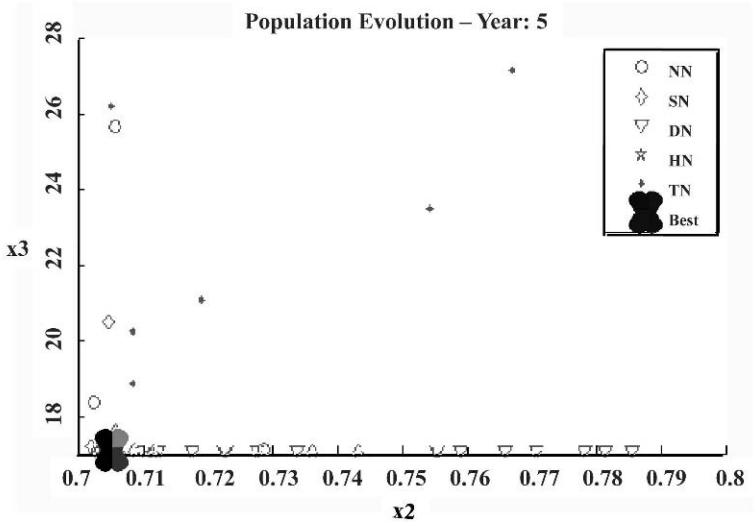


Figure 16-6. Population Swarm Plot of Dimension (x2, x3) at Generation 5.

the other parameters fixed. Table 16-2 describes the parameters in our experimental configuration and the template for our experiments. Each combination of the frequency, configuration, number of hops, and conflict resolution parameter is run 30 times for 1000 generations of a population with 100 individuals.

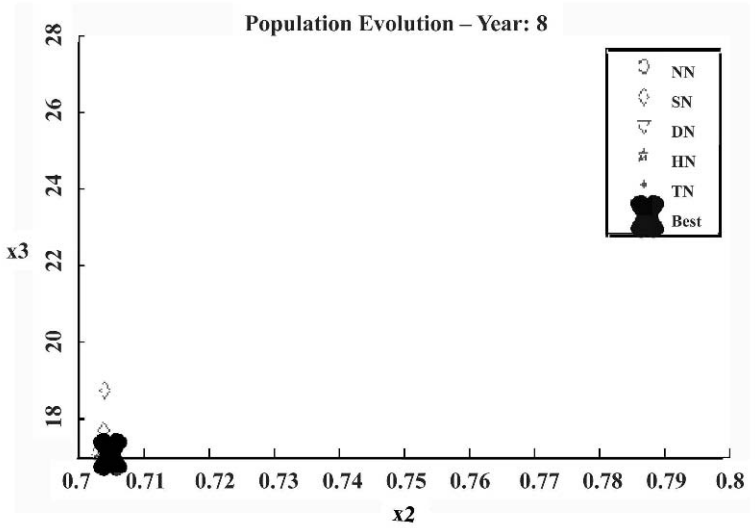


Figure 16-7. Population Swarm Plot of Dimension (x2, x3) at Generation 8.

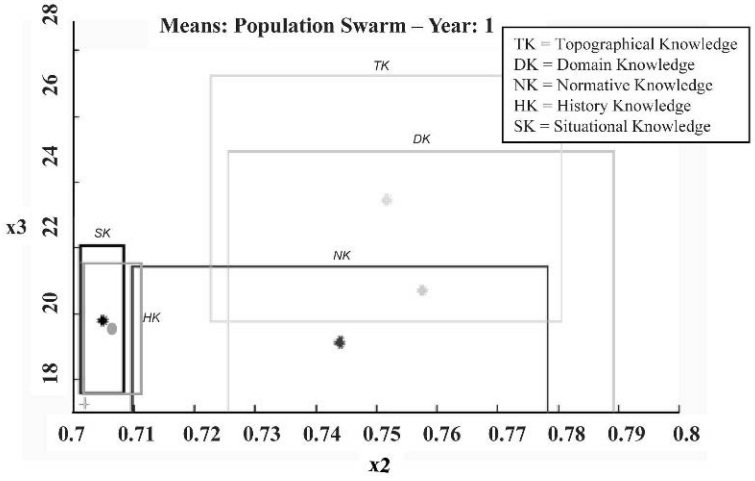


Figure 16-8. Knowledge Swarm Plot of Dimension (x2, x3) at Generation 1.

The averaged "current best" over 1000 generations for each of the 30 runs for our controller, square-direct-3, versus a selected subset of other manually derived configurations is given in Figure 16-12. Included in the subset are two other configurations that are close to ours in terms of variable values, lbest-direct-3 and gbest-direct-3, along with two earlier versions of the Cultural

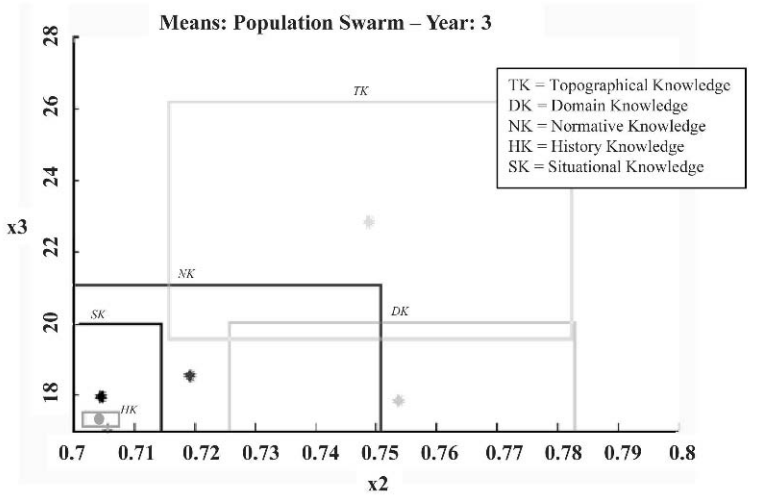


Figure 16-9. Knowledge Swarm Plot of Dimension (x2, x3) at Generation 3.

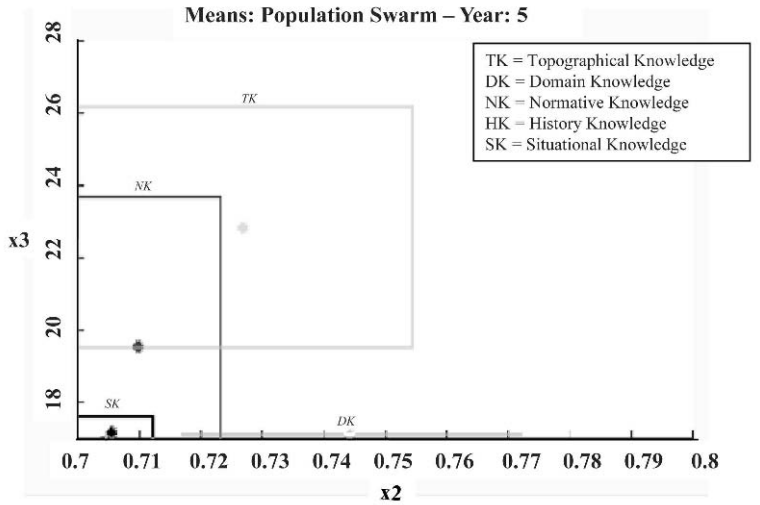


Figure 16-10. Knowledge Swarm Plot of Dimension (x2, x3) at Generation 5.

Algorithm Influence Function without the Social Fabric, MVT (Marginal Value Theorem), and Saleems' (random). Notice that our configuration converges faster to a slightly better solution on average. Table 16-3 compares the different topology-configurations of the SFI using direct resolution, applied every 3 years to the social network. The table shows that the best average weight is obtained using our "Square-direct-3" compared with the other used topologies.

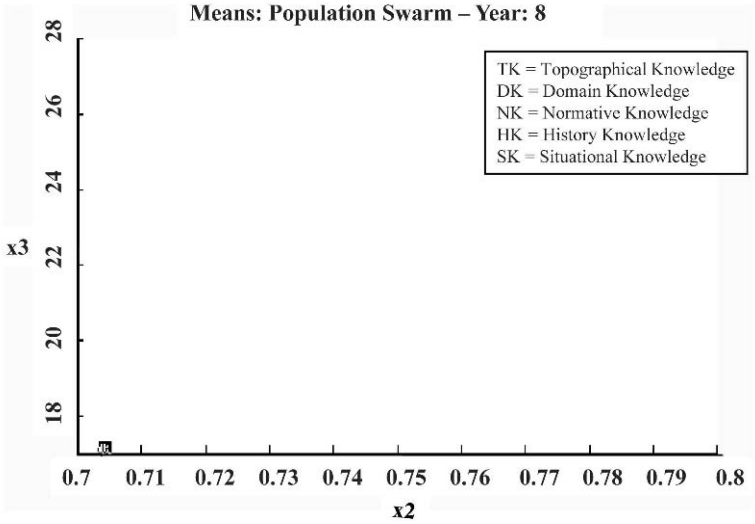


Figure 16-11. Knowledge Swarm Plot of Dimension (x2, x3) at Generation 8.

Table 16-2. Parameters used for Problem P6 using Square Topology.

# of Runs	# of Indiv.	Max. # of Gens.	Freq.	Config. Topology	# of Hops	Conflict Resolution
30	100	1000	1, 3, 8	lBest, Square, gBest	1	Direct, LFU MFU, Rand.

The next question is why does it do better? The answer can be inferred from Table 16-4 where we give the performance of our configuration, SFI(SQUARE), against the two earlier versions of the Cultural Algorithm Influence Function, Random selection (Saleem), and the Marginal Value Theorem (MVT). Notice that by using the parameterized Social Fabric, the search has been focused substantially as indicated by the reduced standard deviation in the social fabric approach. This more focused search produced better statistics across the board with improved "best," "average," and "median" score. And, the worst score is not as bad as the other two as well. Using the derived configuration, the best weight was 2996.974 Kg, the mean weight when we repeated the runs 30 times, 1000 generations each, was 3000.278. The worst value obtained was 3007.301. The gBest approach ranked second with a mean weight of 3011.062 Kg. lBest approach ranked third with mean weight of 3015.026 Kg. The MVT did not perform as good as either of these approaches but is still better than Saleem's random influence function.

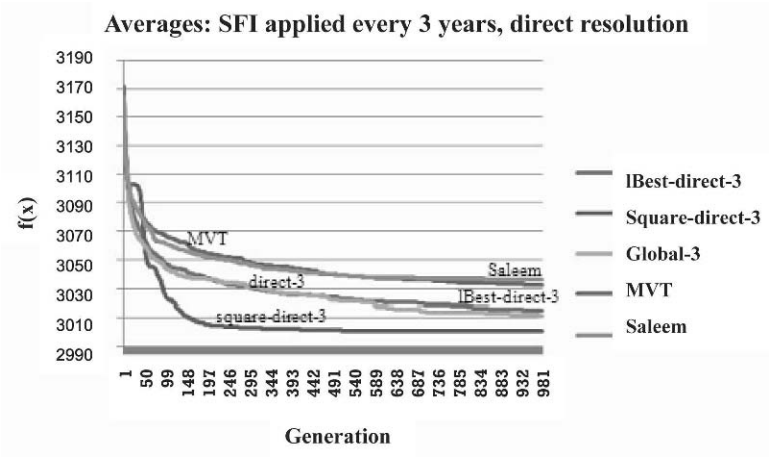


Figure 16-12. "Average-curves" comparison for the derived configuration with the MVT, and Saleem's influence function.

Table 16-3. Statistical comparison of the different topology-configurations of the SFI using direct resolution, applied every 3 years to the social network.

Speed Reducer Design	Statistical Comparison		
	SFI (lBest)	**SFI (Square)**	**SFI (gBest)**
Best	3000.737	2996.974	2998.991
Mean	3015.026	3000.278	3011.062
Median	3013.115	2999.758	3034.973
Worst	3044.332	3007.301	3034.973
St. Dev.	9.949753	2.804042	9.105381

4. Discussion of Results for the Speed Reducer Problem

From the previously presented results, it is clear that the generated configuration for the optimization of Golinski's Speed Reducer problem that "applies" the square topology communication to "spread" the influence of the agents in the social network every 3 generations and "resolves" conflicts between agents using the direct approach outperforms various other parameter combinations of the MVT-Social Fabric Influence Function along with earlier versions of the influence function without the Social Fabric. Also, a comparison with other approaches from the Evolutionary Computation literature shows that performance of the selected configuration compare favorably with them (Coello and Mezura-Montes, 2002). The $(1 + \lambda)$ Evolutionary Strategy produces a mean weight of 3088.778 and took 30,000 generations. The remaining techniques

employed there, such as the Death Penalty and Adaptive approaches, failed to consistently achieve a feasible solution (Coello and Mezura-Montes, 2002). So, while our results are not strictly comparable to theirs since the run parameters are different, the results based upon an incentive-based approach are clearly competitive with them.

Table 16-4. Statistical comparison of the performance of the SFI- best configuration with two earlier version of the CAT Influence Function.

Speed Reducer Design	Statistical Comparison		
	Saleem	**MVT**	**SFI (Square-3-direct**
Best	3020.128	3012.981	2996.974
Mean	3036.156	3032.182	3000.278
Median	3033.782	3029.953	2999.758
Worst	3076.551	3059.519	3007.301
St. Dev.	13.25327	13.06107	2.804042

5. Conclusion

One key question that is frequently raised is, the extent to which Cultural Systems are optimizers? In other words, do the components of a Cultural System facilitate the problem solving process for a group. Here we have set up a Cultural Algorithm framework that allows us to view the effects of social structure on optimization search. In our approach, search is predicated upon how the knowledge sources in the belief space influence the behavior of the population of individuals. The influence can be completely random, or structured based upon a fundamental biological relationship such as predator-prey, or can involve a social network of individuals. The latter case is characteristic of what we associate with in a human Cultural System. So the Influence Function in the CAT environment can viewed as possessing several components. Here we employ the MVT component based on the predator-prey analogy and the Social Fabric component.

So, how does the social configuration of a system reflect, or influence the systems problem solving capabilities? Here we suggest the use of a Genetic Programming method to generate a set of parameter configurations of the Social Fabric to deal with ensembles of different problems. The selected configuration outperformed others in terms of its ability to focus the search more quickly to productive areas without sacrificing the ability to look elsewhere as well, for a well-known but complex Engineering design problem, the Speed Reducer problem. It also compared favorably with results from the optimization literature.

Since only one problem environment was exploited, the tree was a simple condition action rule. However, future work will investigate the use of combinations of problems of varying types, ones that vary from simple to complex. Our ultimate goal is to be able to generate an optimal Social Fabric influence function for combinations of classes of problems of varying complexity, and to see how problem complexity influences the structure of the social networks that emerge in a given Cultural setting.

References

Cheng, L., Patterson, J., Rohall, S., Hupfer, S., and Ross, S. (2005). Weaving a social fabric into existing software aosd 2005. In *Fourth International Conference on Aspect-Oriented Software Development*, Chicago, IL.

Chung, C. and Reynolds, G.R. (1998). Caep: An evolution-based tool for real-valued function optimization using cultural algorithms. *International Journal on Artificial Intelligence Tools*, 7(3):239–291.

Coello, C.A. and Mezura-Montes, E. (2002). Handling constraints in genetic algorithms using dominance-based tournaments. In *Fifth International Conference on Adaptive Computing Design and Manufacture (ACDM)*, volume 5, pages 273–274, University fo Exeter, Devon, UK. Spinger-Verlag.

Golinski, J. (1970). Optimal synthesis problems solved by means of nonlinear programming and random methods. *Journal of Mechanisms*, 5:287–309.

Jin, X. and Reynolds, G.R. (1999). Using knowledge-based evolutionary computation to solve nonlinear constraint optimization problems: a cultural algorithm approach. In *1999 Congress on Evolutionary Computation*, pages 1672–1678, Washington, DC. IEEE.

North, M.J., Collier, N.T., and Vos, J.R. (2006). Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation*, 16(1):1–25.

Ray, T. (2003). Golinski's speed reducer problem revisited. *AIAA J. Technical*, 41(3):556–558.

Reynolds, R.G. (1994). Introduction to cultural algorithms. In Sebald, Anthony V. and Fogel, Lawrence J., editors, *The Third Annual Conference on Evolutionary Programming*, pages 131–139, Singapore. World Scientific Press.

Reynolds, R.G. and Peng, B. (2004). Cultural algorithms: Computational modeling of how cultures learn to solve problems. In *Seventeenth European Meeting on Cybernetics and Systems Research*, Vienna, Austria.

Index